Oracle® Database

Administrator's Guide

10*g* Release 1 (10.1)

Part No. B10739-01

December 2003

ORACLE®

Oracle Database Administrator's Guide, 10*g* Release 1 (10.1)

Part No. B10739-01

Primary Author: Ruth Baylis

Contributing Authors: Paul Lane, Diana Lorentz

Contributors: David Austin, Mark Bauer, Eric Belden, Allen Brumm, Mark Dilman, Harvey Eneman, Amit Ganesh, Carolyn Gray, Joan Gregoire, Daniela Hansell, Wei Huang, Robert Jenkins, Sushil Kumar, Bill Lee, Yunrui Li, Rich Long, Catherine Luu, Mughees Minhas, Valarie Moore, Sujatha Muthulingam, Gary Ngai, Waleed Ojeil, Rod Payne, Ananth Raghavan, Ann Rhee, Jags Srinivasan, Anh-Tuan Tran, Vikarm Shukla, Deborah Steiner, Janet Stern, Michael Stewart, Alex Tsukerman, Kothanda Umamageswaran, Daniel Wong, Wanli Yang

Contents

3 Starting Up and Shutting Down

4 Managing Oracle Database Processes

6 Managing the Redo Log

7 Managing Archived Redo Logs

8 Managing Tablespaces

9 Managing Datafiles and Tempfiles

10 Managing the Undo Tablespace

12 Using Automatic Storage Management

Volume 2

Part IV Schema Objects

13 Managing Space for Schema Objects

14 Managing Tables

17 Managing Clusters

18 Managing Hash Clusters

19 Managing Views, Sequences, and Synonyms

20 General Management of Schema Objects

21 Detecting and Repairing Data Block Corruption

27 Using the Scheduler

28 Administering the Scheduler

30 Managing a Distributed Database

33 Managing Distributed Transactions

Index

Send Us Your Comments

Oracle Database Administrator's Guide 10*g* Release 1 (10.1)

Part No. B10739-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. 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 about this manual?

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

- Electronic mail: infodev_us@oracle.com
- FAX: (650) 506-7227 Attn: Server Technologies Documentation Manager
- Postal service:
  Oracle Corporation
  Server Technologies Documentation
  500 Oracle Parkway, Mailstop 4op11
  Redwood Shores, CA 94065
  USA

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

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

Preface

This guide is for people who administer the operation of an Oracle Database system. Referred to as database administrators (DBAs), they are responsible for creating Oracle Database, ensuring its smooth operation, and monitoring its use.

This preface contains these topics:

Note: The *Oracle Database Administrator's Guide* contains information that describes the features and functionality of the Oracle Database Standard Edition, Oracle Database Enterprise Edition, and Oracle Database Personal Edition products. These products have the same basic features. However, several advanced features are available only with the Oracle Database Enterprise Edition or Oracle Database Personal Edition, and some of these are optional. For example, to create partitioned tables and indexes, you must have the Oracle Database Enterprise Edition or Oracle Database Personal Edition.

For information about the differences between the various editions of Oracle Database and the features and options that are available to you, please refer to *Oracle Database New Features.*

Audience

Readers of this guide are assumed to be familiar with relational database concepts. They are also assumed to be familiar with the operating system environment under which they are running Oracle Database.

Readers Interested in Installation and Upgrade Information

Administrators frequently participate in installing the Oracle Database server software and upgrading an existing Oracle Database to newer formats (for example, Oracle9*i* database to Oracle Database 10*g* format). This guide is not an installation or upgrade manual.

If your primary interest is installation, see your operating system specific Oracle installation guide.

If your primary interest is upgrading a database or application, see the *Oracle Database Upgrade Guide*.

Readers Interested in Application Design Information

In addition to administrators, experienced users of Oracle Database and advanced database application designers might also find information in this guide useful.

However, database application developers should also see the *Oracle Database Application Developer's Guide - Fundamentals* and the documentation for the tool or language product they are using to develop Oracle Database applications.

Organization

This document contains:

Part I, "Basic Database Administration"

Chapter 1, "Overview of Administering an Oracle Database"

This chapter serves as a general introduction to typical tasks performed by database administrators, such as installing software and planning a database.

Chapter 2, "Creating an Oracle Database"

This chapter discusses considerations for creating a database and takes you through the steps of creating one. Consult this chapter when in the database planning and creation stage.

Chapter 3, "Starting Up and Shutting Down"

Consult this chapter when you wish to start up a database, alter its availability, or shut it down. Parameter files related to starting up and shutting down are also described here.

Chapter 4, "Managing Oracle Database Processes"

This chapter helps you to identify different Oracle Database processes, such as dedicated server processes and shared server processes. Consult this chapter when configuring, modifying, tracking and managing processes.

Part II, "Oracle Database Structure and Storage"

Chapter 5, "Managing Control Files"

This chapter describes all aspects of managing control files: naming, creating, troubleshooting, and dropping control files.

Chapter 6, "Managing the Redo Log"

This chapter describes all aspects of managing the online redo log: planning, creating, renaming, dropping, or clearing redo log files.

Chapter 7, "Managing Archived Redo Logs"

Consult this chapter for information about archive modes and tuning archiving.

Chapter 8, "Managing Tablespaces"

This chapter provides guidelines to follow as you manage tablespaces, and describes how to create, manage, alter, drop and move data between tablespaces.

Chapter 9, "Managing Datafiles and Tempfiles"

This chapter provides guidelines to follow as you manage datafiles, and describes how to create, change, alter, rename and view information about datafiles.

Chapter 10, "Managing the Undo Tablespace"

Consult this chapter to learn how to manage undo space using an undo tablespace.

Part III, "Automated File and Storage Management"

Chapter 11, "Using Oracle-Managed Files"

This chapter describes how you can direct the Oracle Database server to create and manage your database files

Chapter 12, "Using Automatic Storage Management"

This chapter briefly discusses some of the concepts behind Automatic Storage Management and describes how to use it.

Part IV, "Schema Objects"

Chapter 13, "Managing Space for Schema Objects"

Consult this chapter for descriptions of common tasks, such as setting storage parameters, deallocating space and managing space.

Chapter 14, "Managing Tables"

Consult this chapter for general table management guidelines, as well as information about creating, altering, maintaining and dropping tables.

Chapter 15, "Managing Indexes"

Consult this chapter for general guidelines about indexes, including creating, altering, monitoring and dropping indexes.

Chapter 16, "Managing Partitioned Tables and Indexes"

Consult this chapter to learn about partitioned tables and indexes and how to create and manage them.

Chapter 17, "Managing Clusters"

Consult this chapter for general guidelines to follow when creating, altering, or dropping clusters.

Chapter 18, "Managing Hash Clusters"

Consult this chapter for general guidelines to follow when creating, altering, or dropping hash clusters.

Chapter 19, "Managing Views, Sequences, and Synonyms"

This chapter describes all aspects of managing views, sequences and synonyms.

Chapter 20, "General Management of Schema Objects"

This chapter covers more varied aspects of schema management. The operations described in this chapter are not unique to any one type of schema objects. Consult this chapter for information about analyzing objects, truncation of tables and clusters, database triggers, integrity constraints, and object dependencies.

Chapter 21, "Detecting and Repairing Data Block Corruption"

This chapter describes methods for detecting and repairing data block corruption.

Part V, "Database Security"

Chapter 22, "Managing Users and Securing the Database"

This chapter discusses the importance of establishing a security policy for your database and users.

Part VI, "Database Resource Management and Task Scheduling"

Chapter 23, "Managing Automatic System Tasks Using the Maintenance Window"

This chapter describes how to take advantage of automatic system tasks.

Chapter 24, "Using the Database Resource Manager"

This chapter describes how to use the Database Resource Manager to allocate resources.

Chapter 25, "Moving from DBMS_JOB to DBMS_SCHEDULER"

This chapter describes how to take statements created with `DBMS_JOB` and rewrite them using `DBMS_SCHEDULER`

Chapter 26, "Overview of Scheduler Concepts"

Oracle Database provides advanced scheduling capabilities through the database Scheduler. This chapter introduces you to its concepts.

Chapter 27, "Using the Scheduler"

This chapter describes how to use the Scheduler.

Chapter 28, "Administering the Scheduler"

This chapter covers the tasks a database administrator needs to perform so end users can schedule jobs using the Scheduler.

Part VII, "Distributed Database Management"

Chapter 29, "Distributed Database Concepts"

This chapter describes the basic concepts and terminology of Oracle Database distributed database architecture.

Chapter 30, "Managing a Distributed Database"

This chapter describes how to manage and maintain a distributed database system.

Chapter 31, "Developing Applications for a Distributed Database System"

This chapter describes considerations important when developing an application to run in a distributed database system.

Chapter 32, "Distributed Transactions Concepts"

This chapter describes what distributed transactions are and how Oracle Database maintains their integrity.

Chapter 33, "Managing Distributed Transactions"

This chapter describes how to manage and troubleshoot distributed transactions.

Related Documentation

For more information, see these Oracle resources:

- *Oracle Database Concepts*

  This book is a starting point to become familiar with the concepts and terminology of the Oracle Database server, and is recommended reading before attempting the tasks described in the *Oracle Database Administrator's Guide*.

- *Oracle Database Backup and Recovery Basics*

  This book provides an overview of backup and recovery and discusses backup and recovery strategies. It provides instructions for basic backup and recovery of your database using Recovery Manager (RMAN).

- *Oracle Database Backup and Recovery Advanced User's Guide*

This guide covers more advanced database backup and recovery topics, including performing user-managed backup and recovery for users who choose not to use RMAN.

- *Oracle Database Performance Tuning Guide*

  This book exposes important considerations in setting up a database system and can help you understand tuning your database. It is mainly conceptual, defining terms, architecture, and design principles, and then outlines proactive and reactive tuning methods.

- *Oracle Database Performance Tuning Guide*

  This book can be used as a reference guide for tuning your Oracle Database system.

- *Oracle Database Application Developer's Guide - Fundamentals*

  Many of the tasks done by DBAs are shared by application developers. In some cases, descriptions of tasks seemed better located in an application level book, and in those cases, this fundamentals book is the primary reference.

Many of the examples in this book use the sample schemas of the seed database, which is installed by default when you install Oracle Database. Refer to *Oracle Database Sample Schemas* for information on how these schemas were created and how you can use them yourself.

Printed documentation is available for sale in the Oracle Store at

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

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

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

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

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

Conventions

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

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.	*Oracle Database Concepts* 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. You can back up the database by using the `BACKUP` command. Query the `TABLE_NAME` column in the `USER_TABLES` data dictionary view. Use the `DBMS_STATS.GENERATE_STATS` procedure.
`lowercase monospace (fixed-width) font`	Lowercase monospace typeface indicates executables, filenames, directory names, and sample user-supplied elements. Such elements include computer and database names, net 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. **Note:** Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.	Enter `sqlplus` to open SQL*Plus. The password is specified in the `orapwd` file. Back up the datafiles and control files in the `/disk1/oracle/dbs` directory. The `department_id`, `department_name`, and `location_id` columns are in the `hr.departments` table. Set the `QUERY_REWRITE_ENABLED` initialization parameter to `true`. Connect as `oe` user. The `JRepUtil` class implements these methods.

Convention	Meaning	Example
lowercase italic monospace (fixed-width) font	Lowercase italic monospace font represents placeholders or variables.	You can specify the *parallel_clause*. Run U*old_release*.SQL where *old_release* refers to the release you installed prior to upgrading.

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}
|	A vertical bar represents a choice of two or more options within brackets or braces. Enter one of the options. Do not enter the vertical bar.	{ENABLE | DISABLE} [COMPRESS | NOCOMPRESS]
...	Horizontal ellipsis points indicate either: ■ That we have omitted parts of the code that are not directly related to the example ■ That you can repeat a portion of the code	CREATE TABLE ... AS *subquery*; SELECT *col1*, *col2*, ... , *coln* FROM employees;

Convention	Meaning	Example
. . .	Vertical ellipsis points indicate that we have omitted several lines of code not directly related to the example.	SQL> SELECT NAME FROM V$DATAFILE; NAME ---------- /fsl/dbs/tbs_01.dbf /fs1/dbs/tbs_02.dbf . . . /fsl/dbs/tbs_09.dbf 9 rows selected.
Other notation	You must enter symbols other than brackets, braces, vertical bars, and ellipsis points as shown.	acctbal NUMBER(11,2); acct CONSTANT NUMBER(4) := 3;
Italics	Italicized text indicates placeholders or variables for which you must supply particular values.	CONNECT SYSTEM/*system_password* 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; SELECT * FROM USER_TABLES; DROP TABLE hr.employees;
lowercase	Lowercase typeface indicates programmatic elements that you supply. For example, lowercase indicates names of tables, columns, or files. **Note:** Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.	SELECT last_name, employee_id FROM employees; sqlplus hr/hr CREATE USER mjones IDENTIFIED BY ty3MU9;

Conventions for Windows Operating Systems

The following table describes conventions for Windows operating systems and provides examples of their use.

Convention	Meaning	Example
Choose Start >	How to start a program.	To start the Database Configuration Assistant, choose Start > Programs > Oracle - *HOME_NAME* > Configuration and Migration Tools > Database Configuration Assistant.
File and directory names	File and directory names are not case sensitive. The following special characters are not allowed: left angle bracket (<), right angle bracket (>), colon (:), double quotation marks ("), slash (/), pipe (|), and dash (-). The special character backslash (\) is treated as an element separator, even when it appears in quotes. If the file name begins with \\, then Windows assumes it uses the Universal Naming Convention.	`c:\winnt"\"system32 is the same as C:\WINNT\SYSTEM32`
`C:\>`	Represents the Windows command prompt of the current hard disk drive. The escape character in a command prompt is the caret (^). Your prompt reflects the subdirectory in which you are working. Referred to as the *command prompt* in this manual.	`C:\oracle\oradata>`
Special characters	The backslash (\) special character is sometimes required as an escape character for the double quotation mark (") special character at the Windows command prompt. Parentheses and the single quotation mark (') do not require an escape character. Refer to your Windows operating system documentation for more information on escape and special characters.	`C:\>exp scott/tiger TABLES=emp QUERY=\"WHERE job='SALESMAN' and sal<1600\"` `C:\>imp SYSTEM/`*`password`*` FROMUSER=scott TABLES=(emp, dept)`
HOME_NAME	Represents the Oracle home name. The home name can be up to 16 alphanumeric characters. The only special character allowed in the home name is the underscore.	`C:\> net start Oracle`*`HOME_NAME`*`TNSListener`

Convention	Meaning	Example
ORACLE_HOME and *ORACLE_BASE*	In releases prior to *Oracle8i* release 8.1.3, when you installed Oracle Database components, all subdirectories were located under a top level *ORACLE_HOME* directory. For Windows NT, the default location was `C:\orant`. This release complies with Optimal Flexible Architecture (OFA) guidelines. All subdirectories are not under a top level *ORACLE_HOME* directory. There is a top level directory called *ORACLE_BASE* that by default is `C:\oracle`. If you install the latest Oracle Database release on a computer with no other Oracle software installed, then the default setting for the first Oracle home directory is `C:\oracle\ora`*nn*, where *nn* is the latest release number. The Oracle home directory is located directly under *ORACLE_BASE*. All directory path examples in this guide follow OFA conventions. Refer to *Oracle Database Platform Guide for Windows* for additional information about OFA compliances and for information about installing Oracle products in non-OFA compliant directories.	Go to the *ORACLE_BASE*`\`*ORACLE_HOME*`\rdbms\admin` directory.

Documentation Accessibility

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

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

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

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

What's New in the Oracle Database 10*g* Administrator's Guide?

This section describes new features of Oracle Database 10*g* Release 1 (10.1) and provides pointers to additional information. New features information from previous releases is retained to help users upgrading to the current release.

For a summary of all new features for Oracle Database 10*g* Release 1 (10.1), see *Oracle Database New Features*. This section describes the new features discussed in the *Oracle Database Administrator's Guide.*

Oracle Database 10*g* Release 1 (10.1) New Features

These are the features introduced in the current release.

- Server manageability features are introduced in "Server Manageability" on page 1-26.
- Automatic system task maintenance is discussed in Chapter 23, "Managing Automatic System Tasks Using the Maintenance Window".
- Bigfile tablespaces

  Oracle Database lets you create single-file tablespaces, called **bigfile tablespaces**, which can contain up to 2^{32} or 4G blocks. The benefits of bigfile tablespaces are the following:

  - They significantly enhance the storage capacity of an Oracle Database.
  - They reduce the number of datafiles needed for an ultra large database.
  - They simplify database management by providing datafile transparency.

See"Supporting Bigfile Tablespaces During Database Creation" on page 2-23 and "Bigfile Tablespaces" on page 8-9.

- Multiple default temporary tablespace support for SQL operations

  You can create a temporary tablespace group that can be specifically assigned to users in the same way that a single temporary tablespace is assigned. A tablespace group can also be specified as the default temporary tablespace for the database.

  See "Multiple Temporary Tablespaces: Using Tablespace Groups" on page 8-21.

- Rename tablespace

  The `RENAME TO` clause of the `ALTER TABLESPACE` statement enables you to rename tablespaces.

  See "Renaming Tablespaces" on page 8-32.

- Cross-platform transportable tablespaces

  Tablespaces can be transported from one platform to another. The RMAN `CONVERT` command is used to do the conversion.

  See "Transporting Tablespaces Between Databases: A Procedure and Example" on page 8-45.

- `SYSAUX` tablespace

  Oracle Database creates an auxiliary system tablespace called `SYSAUX` at database creation. This tablespace can be used by various Oracle Database features and products, rather than saving their data in separate tablespaces or in the `SYSTEM` tablespace.

  See "Creating the SYSAUX Tablespace" on page 2-17 and "Managing the SYSAUX Tablespace" on page 8-34.

- Automatic Storage Management

  Automatic Storage Management provides a logical volume manager integrated with Oracle Database, eliminating the need for you to purchase a third-party product. Oracle Database creates Oracle-managed files within user-defined disk groups that provide redundancy and striping.

  See Chapter 12, "Using Automatic Storage Management".

- Drop database

  The new `DROP DATABASE` statement lets you delete a database and all of its files that are listed in the control file.

See "Dropping a Database" on page 2-45.

- Oracle Flashback Transaction Query

  This feature introduces the FLASHBACK_TRANSACTION_QUERY view, which lets you examine changes to the database at the transaction level. As a result, you can diagnose problems, perform analysis, and audit transactions.

  See "Auditing Table Changes Using Flashback Transaction Query" on page 14-33.

- Oracle Flashback Version Query

  Using undo data stored in the database, you can now view multiple changes to one or more rows, along with the metadata for the changes.

  See "Flashback Features and Undo Space" on page 10-15.

- Oracle Flashback Table

  A new FLASHBACK TABLE statement lets you quickly recover a table to a point in time in the past without restoring a backup.

  See "Recovering Tables Using the Flashback Table Feature" on page 14-34.

- Oracle Flashback Drop

  Oracle Database now provides a way to restore accidentally dropped tables. When tables are dropped, they are placed into a recycle bin from which they can later be recovered.

  See "Using Flashback Drop and Managing the Recycle Bin" on page 14-36.

- Enhanced online redefinition

  New procedures have been added to the DBMS_REDEFINITION package that automate the cloning of dependent objects such as indexes, triggers, privileges, and constraints. Some restrictions have been lifted, allowing more types of tables to be redefined.

  See "Redefining Tables Online" on page 14-24.

- Automatic statistics collection

  You no longer need to specify the MONITORING keyword in the CREATE TABLE or ALTER TABLE statement to enable the automatic collecting of statistics for a table. Statistics are now collected automatically as controlled by the STATISTICS_LEVEL initialization parameter. Automatic statistics collection is the default.

See "Automatically Collecting Statistics on Tables" on page 14-17.

- Scheduler

  Oracle Database provides advanced scheduling capabilities through the database Scheduler.

  See Part VI, "Database Resource Management and Task Scheduling".

- Database Resource Manager enhancement

  The following are enhancements to the Database Resource Manager:

  – Adaptive consumer group mapping

    You can configure the Database Resource Manager to automatically assign consumer groups to sessions by providing mappings between session attributes and consumer groups.

    See "Automatically Assigning Resource Consumer Groups to Sessions" on page 24-27

  – New plan directives

    New resource plan directives let you set idle timeouts, cancel long-running SQL statements, terminate long-running sessions, and restore sessions to their original consumer group at the end of a top call.

    See "Specifying Resource Plan Directives" on page 24-19.

  – New policies

    Two new resource manager policies have also been added: the `RATIO` CPU allocation policy and the `RUN_TO_COMPLETION` scheduling policy.

- New initialization parameter `RESUMABLE_TIMEOUT`

  The `RESUMABLE_TIMEOUT` initialization parameter lets you enable resumable space allocation and set a timeout period across all sessions.

  See "Enabling and Disabling Resumable Space Allocation" on page 13-21.

- Application services

  Tuning by "service and SQL" augments tuning by "session and SQL" in the majority of systems where all sessions are anonymous and shared.

  See "Defining Application Services for Oracle Database 10g" and *Oracle Database Performance Tuning Guide* for more information.

- Simplified recovery through resetlogs

The format for archive log file naming, as specified by the ARCHIVE_LOG_FORMAT initialization parameter, now includes the resetlogs ID, which allows for easier recovery of a database from a previous backup.

See "Specifying Archive Destinations" on page 7-8.

- Automated shared server configuration and simplified shared server configuration parameters.

  You no longer need to preconfigure initialization parameters for shared server. Parameters can be configured dynamically, and most parameters are now limiting parameters to control resources. The recommended method for enabling shared server now is by setting SHARED_SERVERS initialization parameter, rather than the DISPATCHERS initialization parameter.

  See "Configuring Oracle Database for Shared Server" on page 4-5.

- Consolidation of session-specific trace output

  For shared server sessions, the trcsess command-line utility consolidates in one place the trace pertaining to a user session.

  See "Reading the Trace File for Shared Server Sessions" on page 4-31.

- Block remote access to restricted instances

  Remote access to a restricted instance through an Oracle Net listener is blocked.

  See "Restricting Access to an Instance at Startup" on page 3-7.

- Dynamic SGA enhancements

  The JAVA_POOL_SIZE initialization parameter is now dynamic. There is a new STREAMS_POOL_SIZE initialization parameter, which is also dynamic. A new view, V$SGAINFO, provides a consolidated and concise summary of SGA information.

  See "Managing the System Global Area (SGA)" on page 2-32.

- Irreversible database compatibility

  In previous releases you were allowed to lower the compatibility setting for your database. Now, when you advance the compatibility of the database with the COMPATIBLE initialization parameter, you can no longer start the database using a lower compatibility setting, except by doing a point-in-time recovery to a time before the compatibility was advanced.

  See "The COMPATIBLE Initialization Parameter and Irreversible Compatibility" on page 2-43.

- Flash recovery area

  You can create a flash recovery area in your database where Oracle Database can store and manage files related to backup and recovery.

  See "Specifying a Flash Recovery Area" on page 2-29.

- Sorted hash clusters

  Sorted hash clusters are new data structures that allow faster retrieval of data for applications where data is consumed in the order in which it was inserted.

  See "Creating a Sorted Hash Cluster" on page 18-4.

- Copying Files Using the Database Server

  You do not have to use the operating system to copy database files. You can use the `DBMS_FILE_TRANSFER` package to copy files.

  See "Copying Files Using the Database Server" on page 9-15

- Deprecation of `MAXTRANS` physical attribute parameter

  The `MAXTRANS` physical attribute parameter for database objects has been deprecated. Oracle Database now automatically allows up to 255 concurrent update transactions for any data block, depending on the available space in the block.

  See "Specifying the INITRANS Parameter" on page 13-7.

- Deprecation of use of rollback segments (manual undo management mode)

  Manual undo management mode has been deprecated and is no longer documented in this book. Use an undo tablespace and automatic undo management instead.

  See Chapter 10, "Managing the Undo Tablespace".

- Deprecation of the `UNDO_SUPPRESS_ERRORS` initialization parameter

  When operating in automatic undo management mode, the database now ignored any manual undo management mode SQL statements instead of returning error messages.

  See "Overview of Automatic Undo Management" on page 10-2.

- Deprecation of the `PARALLEL_AUTOMATIC_TUNING` initialization parameter

  Oracle Database provides defaults for the parallel execution initialization parameters that are adequate and tuned for most situations. The `PARALLEL_`

AUTOMATIC_TUNING initialization parameter is now redundant and has been deprecated.

- Removal of LogMiner chapter

  The chapter on LogMiner has been moved to *Oracle Database Utilities*

- Removal of job queues chapter on using the DBMS_JOB package

  Using the DBMS_JOB package to submit jobs has been replaced by Scheduler functionality.

  See Part VI, "Database Resource Management and Task Scheduling".

Part I

Basic Database Administration

Part I provides an overview of the responsibilities of a database administrator. This part describes the creation of a database and how to start up and shut down an instance of the database. Part I contains the following chapters:

1

Overview of Administering an Oracle Database

This chapter presents an overview of the environment and tasks of an Oracle Database administrator (DBA). It also discusses DBA security and how you obtain the necessary administrative privileges.

The following topics are discussed:

- Types of Oracle Database Users
- Tasks of a Database Administrator
- Identifying Your Oracle Database Software Release
- Database Administrator Security and Privileges
- Database Administrator Authentication
- Creating and Maintaining a Password File
- Server Manageability

Types of Oracle Database Users

The types of users and their roles and responsibilities depend on the database site. A small site can have one database administrator who administers the database for application developers and users. A very large site can find it necessary to divide the duties of a database administrator among several people and among several areas of specialization.

This section contains the following topics:

- Database Administrators

Database Administrators

Each database requires at least one database administrator (DBA). An Oracle Database system can be large and can have many users. Therefore, database administration is sometimes not a one-person job, but a job for a group of DBAs who share responsibility.

A database administrator's responsibilities can include the following tasks:

- Installing and upgrading the Oracle Database server and application tools
- Allocating system storage and planning future storage requirements for the database system
- Creating primary database storage structures (tablespaces) after application developers have designed an application
- Creating primary objects (tables, views, indexes) once application developers have designed an application
- Modifying the database structure, as necessary, from information given by application developers
- Enrolling users and maintaining system security
- Ensuring compliance with Oracle license agreements
- Controlling and monitoring user access to the database
- Monitoring and optimizing the performance of the database
- Planning for backup and recovery of database information
- Maintaining archived data on tape
- Backing up and restoring the database
- Contacting Oracle for technical support

Security Officers

In some cases, a site assigns one or more security officers to a database. A security officer enrolls users, controls and monitors user access to the database, and maintains system security. As a DBA, you might not be responsible for these duties if your site has a separate security officer. Please refer to *Oracle Database Security Guide* for information about the duties of security officers.

Network Administrators

Some sites have one or more network administrators. A network administrator, for example, administers Oracle networking products, such as Oracle Net Services. Please refer to *Oracle Net Services Administrator's Guide* for information about the duties of network administrators.

> **See Also:** Part VII, "Distributed Database Management", for information on network administration in a distributed environment

Application Developers

Application developers design and implement database applications. Their responsibilities include the following tasks:

- Designing and developing the database application
- Designing the database structure for an application
- Estimating storage requirements for an application
- Specifying modifications of the database structure for an application
- Relaying this information to a database administrator
- Tuning the application during development
- Establishing security measures for an application during development

Application developers can perform some of these tasks in collaboration with DBAs. Please refer to *Oracle Database Application Developer's Guide - Fundamentals* for information about application development tasks.

Application Administrators

An Oracle Database site can assign one or more application administrators to administer a particular application. Each application can have its own administrator.

Database Users

Database users interact with the database through applications or utilities. A typical user's responsibilities include the following tasks:

- Entering, modifying, and deleting data, where permitted
- Generating reports from the data

Tasks of a Database Administrator

The following tasks present a prioritized approach for designing, implementing, and maintaining an Oracle Database:

Task 1: Evaluate the Database Server Hardware

Task 2: Install the Oracle Database Software

Task 3: Plan the Database

Task 4: Create and Open the Database

Task 5: Back Up the Database

Task 6: Enroll System Users

Task 7: Implement the Database Design

Task 8: Back Up the Fully Functional Database

Task 9: Tune Database Performance

These tasks are discussed in the sections that follow.

Note: When upgrading to a new release, back up your existing production environment, both software and database, before installation. For information on preserving your existing production database, see *Oracle Database Upgrade Guide*.

Task 1: Evaluate the Database Server Hardware

Evaluate how Oracle Database and its applications can best use the available computer resources. This evaluation should reveal the following information:

- How many disk drives are available to the Oracle products
- How many, if any, dedicated tape drives are available to Oracle products
- How much memory is available to the instances of Oracle Database you will run (see your system configuration documentation)

Task 2: Install the Oracle Database Software

As the database administrator, you install the Oracle Database server software and any front-end tools and database applications that access the database. In some distributed processing installations, the database is controlled by a central computer (database server) and the database tools and applications are executed on remote computers (clients). In this case, you must also install the Oracle Net components necessary to connect the remote machines to the computer that executes Oracle Database.

For more information on what software to install, see "Identifying Your Oracle Database Software Release" on page 1-8.

> **See Also:** For specific requirements and instructions for installation, refer to the following documentation:
>
> - The Oracle documentation specific to your operating system
> - The installation guides for your front-end tools and Oracle Net drivers

Task 3: Plan the Database

As the database administrator, you must plan:

- The logical storage structure of the database
- The overall database design
- A backup strategy for the database

It is important to plan how the logical storage structure of the database will affect system performance and various database management operations. For example, before creating any tablespaces for your database, you should know how many datafiles will make up the tablespace, what type of information will be stored in

each tablespace, and on which disk drives the datafiles will be physically stored. When planning the overall logical storage of the database structure, take into account the effects that this structure will have when the database is actually created and running. Consider how the logical storage structure of the database will affect:

- The performance of the computer executing running Oracle Database
- The performance of the database during data access operations
- The efficiency of backup and recovery procedures for the database

Plan the relational design of the database objects and the storage characteristics for each of these objects. By planning the relationship between each object and its physical storage before creating it, you can directly affect the performance of the database as a unit. Be sure to plan for the growth of the database.

In distributed database environments, this planning stage is extremely important. The physical location of frequently accessed data dramatically affects application performance.

During the planning stage, develop a backup strategy for the database. You can alter the logical storage structure or design of the database to improve backup efficiency.

It is beyond the scope of this book to discuss relational and distributed database design. If you are not familiar with such design issues, please refer to accepted industry-standard documentation.

Part II, "Oracle Database Structure and Storage", and Part IV, "Schema Objects", provide specific information on creating logical storage structures, objects, and integrity constraints for your database.

Task 4: Create and Open the Database

After you complete the database design, you can create the database and open it for normal use. You can create a database at installation time, using the Database Configuration Assistant, or you can supply your own scripts for creating a database.

Please refer to Chapter 2, "Creating an Oracle Database", for information on creating a database and Chapter 3, "Starting Up and Shutting Down" for guidance in starting up the database.

Task 5: Back Up the Database

After you create the database structure, carry out the backup strategy you planned for the database. Create any additional redo log files, take the first full database backup (online or offline), and schedule future database backups at regular intervals.

See Also:

- *Oracle Database Backup and Recovery Basics*
- *Oracle Database Backup and Recovery Advanced User's Guide*

Task 6: Enroll System Users

After you back up the database structure, you can enroll the users of the database in accordance with your Oracle license agreement, and grant appropriate privileges and roles to these users. Please refer to Chapter 22, "Managing Users and Securing the Database" for guidance in this task.

Task 7: Implement the Database Design

After you create and start the database, and enroll the system users, you can implement the planned logical structure database by creating all necessary tablespaces. When you have finished creating tablespaces, you can create the database objects.

Part II, "Oracle Database Structure and Storage" and Part IV, "Schema Objects" provide information on creating logical storage structures and objects for your database.

Task 8: Back Up the Fully Functional Database

When the database is fully implemented, again back up the database. In addition to regularly scheduled backups, you should always back up your database immediately after implementing changes to the database structure.

Task 9: Tune Database Performance

Optimizing the performance of the database is one of your ongoing responsibilities as a DBA. Oracle Database provides a database resource management feature that helps you to control the allocation of resources among various user groups. The database resource manager is described in Chapter 24, "Using the Database Resource Manager".

See Also: *Oracle Database Performance Tuning Guide* for information about tuning your database and applications

Identifying Your Oracle Database Software Release

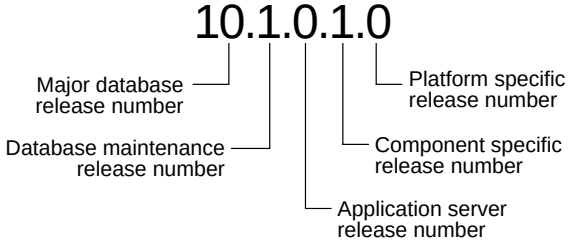
Because Oracle Database continues to evolve and can require maintenance, Oracle periodically produces new releases. Not all customers initially subscribe to a new release or require specific maintenance for their existing release. As a result, multiple releases of the product exist simultaneously.

As many as five numbers may be required to fully identify a release. The significance of these numbers is discussed in the sections that follow.

Release Number Format

To understand the release nomenclature used by Oracle, examine the following example of an Oracle Database server labeled "Release 10.1.0.1.0".

Figure 1–1 Example of an Oracle Database Release Number



Note: Starting with release 9.2, maintenance releases of Oracle Database are denoted by a change to the second digit of a release number. In previous releases, the third digit indicated a particular maintenance release.

Major Database Release Number

The first digit is the most general identifier. It represents a major new version of the software that contains significant new functionality.

Database Maintenance Release Number

The second digit represents a maintenance release level. Some new features may also be included.

Application Server Release Number

The third digit reflects the release level of the Oracle Application Server (OracleAS).

Component-Specific Release Number

The fourth digit identifies a release level specific to a component. Different components can have different numbers in this position depending upon, for example, component patch sets or interim releases.

Platform-Specific Release Number

The fifth digit identifies a platform-specific release. Usually this is a patch set. When different platforms require the equivalent patch set, this digit will be the same across the affected platforms.

Checking Your Current Release Number

To identify the release of Oracle Database that is currently installed and to see the release levels of other database components you are using, query the data dictionary view PRODUCT_COMPONENT_VERSION. A sample query follows. (You can also query the V$VERSION view to see component-level information.) Other product release levels may increment independent of the database server.

```
COL PRODUCT FORMAT A35
COL VERSION FORMAT A15
COL STATUS FORMAT A15
SELECT * FROM PRODUCT_COMPONENT_VERSION;

PRODUCT                                    VERSION     STATUS
------------------------------------------ ----------- -----------
NLSRTL                                     10.1.0.2.0  Production
Oracle Database 10g Enterprise Edition     10.1.0.2.0  Prod
PL/SQL                                     10.1.0.2.0  Production
...
```

It is important to convey to Oracle the results of this query when you report problems with the software.

Database Administrator Security and Privileges

To perform the administrative tasks of an Oracle Database DBA, you need specific privileges within the database and possibly in the operating system of the server on which the database runs. Access to a database administrator's account should be tightly controlled.

This section contains the following topics:

- The Database Administrator's Operating System Account
- Database Administrator Usernames

The Database Administrator's Operating System Account

To perform many of the administrative duties for a database, you must be able to execute operating system commands. Depending on the operating system on which Oracle Database is running, you might need an operating system account or ID to gain access to the operating system. If so, your operating system account might require operating system privileges or access rights that other database users do not require (for example, to perform Oracle Database software installation). Although you do not need the Oracle Database files to be stored in your account, you should have access to them.

> **See Also:** Your operating system specific Oracle documentation. The method of creating the account of the database administrator is specific to the operating system.

Database Administrator Usernames

Two user accounts are automatically created when Oracle Database is installed:

- SYS (default password: CHANGE_ON_INSTALL)
- SYSTEM (default password: MANAGER)

Note: Both Oracle Universal Installer (OUI) and Database Configuration Assistant (DBCA) now prompt for SYS and SYSTEM passwords and do not accept the default passwords "change_on_ install" or "manager", respectively.

If you create the database manually, Oracle strongly recommends that you specify passwords for SYS and SYSTEM at database creation time, rather than using these default passwords. Please refer to "Protecting Your Database: Specifying Passwords for Users SYS and SYSTEM" on page 2-15 for more information.

Create at least one additional administrative user and grant to that user an appropriate administrative role to use when performing daily administrative tasks. Do not use SYS and SYSTEM for these purposes.

Note Regarding Security Enhancements: In this release of Oracle Database and in subsequent releases, several enhancements are being made to ensure the security of default database user accounts. You can find a security checklist for this release in *Oracle Database Security Guide*. Oracle recommends that you read this checklist and configure your database accordingly.

SYS

When you create an Oracle Database, the user SYS is automatically created and granted the DBA role.

All of the base tables and views for the database data dictionary are stored in the schema SYS. These base tables and views are critical for the operation of Oracle Database. To maintain the integrity of the data dictionary, tables in the SYS schema are manipulated only by the database. They should never be modified by any user or database administrator, and no one should create any tables in the schema of user SYS. (However, you can change the storage parameters of the data dictionary settings if necessary.)

Ensure that most database users are never able to connect to Oracle Database using the SYS account.

SYSTEM

When you create an Oracle Database, the user SYSTEM is also automatically created and granted the DBA role.

The SYSTEM username is used to create additional tables and views that display administrative information, and internal tables and views used by various Oracle Database options and tools. Never use the SYSTEM schema to store tables of interest to nonadministrative users.

The DBA Role

A predefined DBA role is automatically created with every Oracle Database installation. This role contains most database system privileges. Therefore, the DBA role should be granted only to actual database administrators.

Note: The DBA role does not include the SYSDBA or SYSOPER system privileges. These are special administrative privileges that allow an administrator to perform basic database administration tasks, such as creating the database and instance startup and shutdown. These system privileges are discussed in "Administrative Privileges" on page 1-12.

Database Administrator Authentication

As a DBA, you often perform special operations such as shutting down or starting up a database. Because only a DBA should perform these operations, the database administrator usernames require a secure authentication scheme.

This section contains the following topics:

- Administrative Privileges
- Selecting an Authentication Method
- Using Operating System Authentication
- Using Password File Authentication

Administrative Privileges

Administrative privileges that are required for an administrator to perform basic database operations are granted through two special system privileges, SYSDBA

and SYSOPER. You must have one of these privileges granted to you, depending upon the level of authorization you require.

> **Note:** The SYSDBA and SYSOPER system privileges allow access to a database instance even when the database is not open. Control of these privileges is totally outside of the database itself.
>
> The SYSDBA and SYSOPER privileges can also be thought of as types of connections that enable you to perform certain database operations for which privileges cannot be granted in any other fashion. For example, you if you have the SYSDBA privilege, you can connect to the database by specifying `CONNECT AS SYSDBA`.

SYSDBA and SYSOPER

The following operations are authorized by the SYSDBA and SYSOPER system privileges:

System Privilege	Operations Authorized
SYSDBA	■ Perform `STARTUP` and `SHUTDOWN` operations ■ `ALTER DATABASE`: open, mount, back up, or change character set ■ `CREATE DATABASE` ■ `DROP DATABASE` ■ `CREATE SPFILE` ■ `ALTER DATABASE ARCHIVELOG` ■ `ALTER DATABASE RECOVER` ■ Includes the `RESTRICTED SESSION` privilege Effectively, this system privilege allows a user to connect as user SYS.

System Privilege	Operations Authorized
SYSOPER	■ Perform STARTUP and SHUTDOWN operations ■ CREATE SPFILE ■ ALTER DATABASE OPEN/MOUNT/BACKUP ■ ALTER DATABASE ARCHIVELOG ■ ALTER DATABASE RECOVER (Complete recovery only. Any form of incomplete recovery, such as UNTIL TIME|CHANGE|CANCEL|CONTROLFILE requires connecting as SYSDBA.) ■ Includes the RESTRICTED SESSION privilege This privilege allows a user to perform basic operational tasks, but without the ability to look at user data.

The manner in which you are authorized to use these privileges depends upon the method of authentication that you use.

When you connect with SYSDBA or SYSOPER privileges, you connect with a default schema, not with the schema that is generally associated with your username. For SYSDBA this schema is SYS; for SYSOPER the schema is PUBLIC.

Connecting with Administrative Privileges: Example

This example illustrates that a user is assigned another schema (SYS) when connecting with the SYSDBA system privilege. Assume that the sample user oe has been granted the SYSDBA system privilege and has issued the following statements:

```
CONNECT oe/oe
CREATE TABLE admin_test(name VARCHAR2(20));
```

Later, user oe issues these statements:

```
CONNECT oe/oe AS SYSDBA
SELECT * FROM admin_test;
```

User oe now receives the following error:

```
ORA-00942: table or view does not exist
```

Having connected as SYSDBA, user oe now references the SYS schema, but the table was created in the oe schema.

See Also:

- "Using Operating System Authentication" on page 1-17
- "Using Password File Authentication" on page 1-18

Selecting an Authentication Method

The following methods are available for authenticating database administrators:

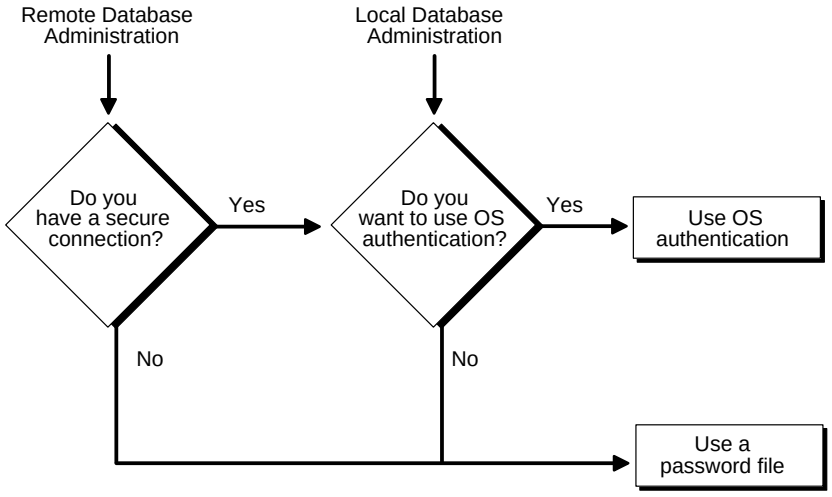
- Operating system (OS) authentication
- A password file

Notes:

- These methods replace the `CONNECT INTERNAL` syntax provided with earlier versions of Oracle Database. `CONNECT INTERNAL` is no longer supported.
- Operating system authentication takes precedence over password file authentication. If you meet the requirements for operating system authentication, then even if you use a password file, you will be authenticated by operating system authentication.

Your choice will be influenced by whether you intend to administer your database locally on the same machine where the database resides, or whether you intend to administer many different databases from a single remote client. Figure 1–2 illustrates the choices you have for database administrator authentication schemes.

Figure 1–2 Database Administrator Authentication Methods



If you are performing remote database administration, consult your Oracle Net documentation to determine whether you are using a secure connection. Most popular connection protocols, such as TCP/IP and DECnet, are not secure.

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

Nonsecure Remote Connections

To connect to Oracle Database as a privileged user over a nonsecure connection, you must be authenticated by a password file. When using password file authentication, the database uses a password file to keep track of database usernames that have been granted the SYSDBA or SYSOPER system privilege. This form of authentication is discussed in "Using Password File Authentication" on page 1-18.

Local Connections and Secure Remote Connections

You can connect to Oracle Database as a privileged user over a local connection or a secure remote connection in two ways:

- If the database has a password file and you have been granted the SYSDBA or SYSOPER system privilege, then you can connect and be authenticated by a password file.

- If the server is not using a password file, or if you have not been granted SYSDBA or SYSOPER privileges and are therefore not in the password file, you can use operating system authentication. On most operating systems, authentication for database administrators involves placing the operating system username of the database administrator in a special group, generically referred to as OSDBA. Users in that group are granted SYSDBA privileges. A similar group, OSOPER, is used to grant SYSOPER privileges to users.

Using Operating System Authentication

This section describes how to authenticate an administrator using the operating system.

Preparing to Use Operating System Authentication

To enable operating system authentication of an administrative user:

1. Create an operating system account for the user.
2. Add the user to the OSDBA or OSOPER operating system defined groups.
3. Ensure that the initialization parameter, REMOTE_LOGIN_PASSWORDFILE, is set to NONE, the default.

Connecting Using Operating System Authentication

A user can be authenticated, enabled as an administrative user, and connected to a local database by typing one of the following SQL*Plus commands:

```
CONNECT / AS SYSDBA
CONNECT / AS SYSOPER
```

For a remote database connection over a secure connection, the user must also specify the net service name of the remote database:

```
CONNECT /@net_service_name AS SYSDBA
CONNECT /@net_service_name AS SYSOPER
```

> **See Also:** *SQL*Plus User's Guide and Reference* for syntax of the CONNECT command

OSDBA and OSOPER

Two special operating system groups control database administrator connections when using operating system authentication. These groups are generically referred to as OSDBA and OSOPER. The groups are created and assigned specific names as

part of the database installation process. The specific names vary depending upon your operating system and are listed in the following table:

Operating System Group	UNIX	Windows
OSDBA	dba	ORA_DBA
OSOPER	oper	ORA_OPER

The default names assumed by the Oracle Universal Installer can be overridden. How you create the OSDBA and OSOPER groups is operating system specific.

Membership in the OSDBA or OSOPER group affects your connection to the database in the following ways:

- If you are a member of the OSDBA group and you specify AS SYSDBA when you connect to the database, then you connect to the database with the SYSDBA system privilege.
- If you are a member of the OSOPER group and you specify AS SYSOPER when you connect to the database, then you connect to the database with the SYSOPER system privilege.
- If you are not a member of either of these operating system groups, the CONNECT command fails.

> **See Also:** Your operating system specific Oracle documentation for information about creating the OSDBA and OSOPER groups

Using Password File Authentication

This section describes how to authenticate an administrative user using password file authentication.

Preparing to Use Password File Authentication

To enable authentication of an administrative user using password file authentication you must do the following:

1. Create an operating system account for the user.
2. If not already created, create the password file using the ORAPWD utility:

```
ORAPWD FILE=filename PASSWORD=password ENTRIES=max_users
```

3. Set the REMOTE_LOGIN_PASSWORDFILE initialization parameter to EXCLUSIVE.
4. Connect to the database as user SYS (or as another user with the administrative privileges).
5. If the user does not already exist in the database, create the user. Grant the SYSDBA or SYSOPER system privilege to the user:

   ```
   GRANT SYSDBA to oe;
   ```

   This statement adds the user to the password file, thereby enabling connection AS SYSDBA.

   See Also: "Creating and Maintaining a Password File" on page 1-20 for instructions for creating and maintaining a password file

Connecting Using Password File Authentication

Administrative users can be connected and authenticated to a local or remote database by using the SQL*Plus CONNECT command. They must connect using their username and password and the AS SYSDBA or AS SYSOPER clause. For example, user oe has been granted the SYSDBA privilege, so oe can connect as follows:

```
CONNECT oe/oe AS SYSDBA
```

However, user oe has not been granted the SYSOPER privilege, so the following command will fail:

```
CONNECT oe/oe AS SYSOPER
```

Note: Operating system authentication takes precedence over password file authentication. Specifically, if you are a member of the OSDBA or OSOPER group for the operating system, and you connect as SYSDBA or SYSOPER, you will be connected with associated administrative privileges regardless of the *username/password* that you specify.

If you are not in the OSDBA or OSOPER groups, and you are not in the password file, then the connection will fail.

See Also: *SQL*Plus User's Guide and Reference* for syntax of the CONNECT command

Creating and Maintaining a Password File

You can create a password file using the password file creation utility, ORAPWD. For some operating systems, you can create this file as part of your standard installation.

This section contains the following topics:

Using ORAPWD

When you invoke this password file creation utility without supplying any parameters, you receive a message indicating the proper use of the command as shown in the following sample output:

```
orapwd
Usage: orapwd file=<fname> password=<password> entries=<users>
where
file - name of password file (mand),
password - password for SYS (mand),
entries - maximum number of distinct DBAs and OPERs (opt),
There are no spaces around the equal-to (=) character.
```

The following command creates a password file named `acct.pwd` that allows up to 30 privileged users with different passwords. In this example, the file is initially created with the password `secret` for users connecting as SYS.

```
orapwd FILE=acct.pwd PASSWORD=secret ENTRIES=30
```

The parameters in the ORAPWD utility are described in the sections that follow.

FILE

This parameter sets the name of the password file being created. You must specify the full path name for the file. The contents of this file are encrypted, and the file cannot be read directly. This parameter is mandatory.

The types of filenames allowed for the password file are operating system specific. Some operating systems require the password file to adhere to a specific format and be located in a specific directory. Other operating systems allow the use of

environment variables to specify the name and location of the password file. See your operating system documentation for the names and locations allowed on your platform.

If you are running multiple instances of Oracle Database using Oracle Real Application Clusters, the environment variable for each instance should point to the same password file.

Caution: It is critically important to the security of your system that you protect your password file and the environment variables that identify the location of the password file. Any user with access to these could potentially compromise the security of the connection.

PASSWORD

This parameter sets the password for user SYS. If you issue the ALTER USER statement to change the password for SYS after connecting to the database, both the password stored in the data dictionary and the password stored in the password file are updated. This parameter is mandatory.

Note: You cannot change the password for SYS if REMOTE_LOGON_PASSWORDFILE is set to SHARED. An error message is issued if you attempt to do so.

ENTRIES

This parameter specifies the number of entries that you require the password file to accept. This number corresponds to the number of distinct users allowed to connect to the database as SYSDBA or SYSOPER. The actual number of allowable entries can be higher than the number of users, because the ORAPWD utility continues to assign password entries until an operating system block is filled. For example, if your operating system block size is 512 bytes, it holds four password entries. The number of password entries allocated is always a multiple of four.

Entries can be reused as users are added to and removed from the password file. If you intend to specify REMOTE_LOGON_PASSWORDFILE=EXCLUSIVE, and to allow the granting of SYSDBA and SYSOPER privileges to users, this parameter is required.

Caution: When you exceed the allocated number of password entries, you must create a new password file. To avoid this necessity, allocate a number of entries that is larger than you think you will ever need.

Setting REMOTE_LOGIN_ PASSWORDFILE

In addition to creating the password file, you must also set the initialization parameter REMOTE_LOGIN_PASSWORDFILE to the appropriate value. The values recognized are:

- NONE: Setting this parameter to NONE causes Oracle Database to behave as if the password file does not exist. That is, no privileged connections are allowed over nonsecure connections. NONE is the default value for this parameter.
- EXCLUSIVE: An EXCLUSIVE password file can be used with only one database. Only an EXCLUSIVE file can contain the names of users other than SYS. Using an EXCLUSIVE password file lets you grant SYSDBA and SYSOPER system privileges to individual users and have them connect as themselves.
- SHARED: A SHARED password file can be used by multiple databases running on the same server. However, the only user recognized by a SHARED password file is SYS. You cannot add users to a SHARED password file. All users needing SYSDBA or SYSOPER system privileges must connect using the same name, SYS, and password. This option is useful if you have a single DBA administering multiple databases.

Suggestion: To achieve the greatest level of security, you should set the REMOTE_LOGIN_PASSWORDFILE initialization parameter to EXCLUSIVE immediately after creating the password file.

Adding Users to a Password File

When you grant SYSDBA or SYSOPER privileges to a user, that user's name and privilege information are added to the password file. If the server does not have an EXCLUSIVE password file (that is, if the initialization parameter REMOTE_LOGIN_ PASSWORDFILE is NONE or SHARED) Oracle Database issue an error if you attempt to grant these privileges.

A user's name remains in the password file only as long as that user has at least one of these two privileges. If you revoke both of these privileges, Oracle Database removes the user from the password file.

Creating a Password File and Adding New Users to It

Use the following procedure to create a password and add new users to it:

1. Follow the instructions for creating a password file as explained in "Using ORAPWD" on page 1-20.
2. Set the REMOTE_LOGIN_PASSWORDFILE initialization parameter to EXCLUSIVE.
3. Connect with SYSDBA privileges as shown in the following example:

   ```
   CONNECT SYS/password AS SYSDBA
   ```

4. Start up the instance and create the database if necessary, or mount and open an existing database.
5. Create users as necessary. Grant SYSDBA or SYSOPER privileges to yourself and other users as appropriate. See "Granting and Revoking SYSDBA and SYSOPER Privileges".

Granting the SYSDBA or SYSOPER privilege to a user causes that user's username to be added to the password file. This enables the user to connect to the database as SYSDBA or SYSOPER by specifying username and password (instead of using SYS). The use of a password file does not prevent operating system authenticated users from connecting if they meet the criteria for operating system authentication.

Granting and Revoking SYSDBA and SYSOPER Privileges

If your server is using an EXCLUSIVE password file, use the GRANT statement to grant the SYSDBA or SYSOPER system privilege to a user, as shown in the following example:

```
GRANT SYSDBA TO oe;
```

Use the REVOKE statement to revoke the SYSDBA or SYSOPER system privilege from a user, as shown in the following example:

```
REVOKE SYSDBA FROM oe;
```

Because SYSDBA and SYSOPER are the most powerful database privileges, the WITH ADMIN OPTION is not used in the GRANT statement. That is, the grantee cannot in turn grant the SYSDBA or SYSOPER privilege to another user. Only a user currently

connected as SYSDBA can grant or revoke another user's SYSDBA or SYSOPER system privileges. These privileges cannot be granted to roles, because roles are available only after database startup. Do not confuse the SYSDBA and SYSOPER database privileges with operating system roles.

> **See Also:** *Oracle Database Security Guide* for more information on system privileges

Viewing Password File Members

Use the V$PWFILE_USERS view to see the users who have been granted SYSDBA or SYSOPER system privileges for a database. The columns displayed by this view are as follows:

Column	Description
USERNAME	This column contains the name of the user that is recognized by the password file.
SYSDBA	If the value of this column is TRUE, then the user can log on with SYSDBA system privileges.
SYSOPER	If the value of this column is TRUE, then the user can log on with SYSOPER system privileges.

Maintaining a Password File

This section describes how to:

- Expand the number of password file users if the password file becomes full
- Remove the password file
- Avoid changing the state of the password file

Expanding the Number of Password File Users

If you receive the file full error (ORA-1996) when you try to grant SYSDBA or SYSOPER system privileges to a user, you must create a larger password file and regrant the privileges to the users.

Replacing a Password File

Use the following procedure to replace a password file:

1. Identify the users who have SYSDBA or SYSOPER privileges by querying the V$PWFILE_USERS view.

2. Shut down the database.
3. Delete the existing password file.
4. Follow the instructions for creating a new password file using the ORAPWD utility in "Using ORAPWD" on page 1-20. Ensure that the ENTRIES parameter is set to a number larger than you think you will ever need.
5. Follow the instructions in "Adding Users to a Password File" on page 1-22.

Removing a Password File

If you determine that you no longer require a password file to authenticate users, you can delete the password file and reset the REMOTE_LOGIN_PASSWORDFILE initialization parameter to NONE. After you remove this file, only those users who can be authenticated by the operating system can perform database administration operations.

> **Caution:** Do not remove or modify the password file if you have a database or instance mounted using REMOTE_LOGIN_PASSWORDFILE=EXCLUSIVE (or SHARED). If you do, you will be unable to reconnect remotely using the password file. Even if you replace it, you cannot use the new password file, because the timestamps and checksums will be wrong.

Changing the Password File State

The password file state is stored in the password file. When you first create a password file, its default state is SHARED. You can change the state of the password file by setting the initialization parameter REMOTE_LOGIN_PASSWORDFILE. When you start up an instance, Oracle Database retrieves the value of this parameter from the parameter file used by the instance. When you mount the database, the database compares the value of this parameter to the value stored in the password file. If the values do not match, the database overwrites the value stored in the file.

Caution: Ensure that an EXCLUSIVE password file is not accidentally changed to SHARED. If you plan to allow instance startup from multiple clients, each of those clients must have an initialization parameter file, and the value of the parameter REMOTE_LOGIN_PASSWORDFILE must be the same in each of these files. Otherwise, the state of the password file could change depending upon where the instance was started.

Server Manageability

Oracle Database is a sophisticated self-managing database that automatically monitors, adapts, and repairs itself. It automates routine DBA tasks and reduces the complexity of space, memory, and resource administration. Several advisors are provided to help you analyze specific objects. The advisors report on a variety of aspects of the object and describe recommended actions. Oracle Database proactively sends alerts when a problem is anticipated or when any of the user-selected metrics.

In addition to its self-managing features, Oracle Database provides utilities to help you move data in and out of the database.

This section describes these server manageability topics:

- Automatic Manageability Features
- Data Utilities

Automatic Manageability Features

Oracle Database has a self-management infrastructure that allows the database to learn about itself and use this information to adapt to workload variations and to automatically remedy any potential problem. This section discusses the automatic manageability features of Oracle Database.

Automatic Workload Repository

Automatic Workload Repository (AWR) is a built-in repository in every Oracle Database. At regular intervals, the database makes a snapshot of all its vital statistics and workload information and stores them in AWR. By default, the snapshots are made every 30 minutes, but you can change this frequency. The snapshots are stored in the AWR for a period of time (seven days by default) after which they are automatically purged.

See Also:

- *Oracle Database Concepts* and *Oracle 2 Day DBA* for overviews the repository
- *Oracle Database Performance Tuning Guide* for details on using Automatic Workload Repository for statistics collection

Automatic Maintenance Tasks

Oracle Database uses the information stored in AWR to identify the need to perform routine maintenance tasks, such as optimizer statistics refresh and rebuilding indexes. Then the database uses the Scheduler to run such tasks in a predefined maintenance window.

See Also:

- *Oracle Database Concepts* for an overview of the maintenance window
- Chapter 23, "Managing Automatic System Tasks Using the Maintenance Window" for detailed information on using the predefined maintenance window

Server-Generated Alerts

Some problems cannot be resolved automatically and require the database administrator's attention. For these problems, such as space shortage, Oracle Database provides server-generated alerts to notify you when then problem arises. The alerts also provide recommendations on how to resolve the problem.

See Also: "Server-Generated Alerts" on page 4-25 in this book for detailed information on using APIs to administer server-generated alerts

Advisors

Oracle Database provides advisors to help you optimize a number of subsystems in the database. An advisory framework ensures consistency in the way in which advisors are invoked and results are reported. The advisors are use primarily by the database to optimize its own performance. However, they can also be invoked by administrators to get more insight into the functioning of a particular subcomponent.

See Also:

- *Oracle Database Concepts* for an overview of the advisors
- *Oracle 2 Day DBA* for more detailed information on using the advisors

Data Utilities

Several utilities are available to help you maintain the data in your Oracle Database. This section introduces two of these utilities:

- SQL*Loader
- Export and Import Utilities

See Also: *Oracle Database Utilities* for detailed information about these utilities

SQL*Loader

SQL*Loader is used both by database administrators and by other users of Oracle Database. It loads data from standard operating system files (such as, files in text or C data format) into database tables.

Export and Import Utilities

Oracle export and import utilities enable you to move existing data in Oracle format between one Oracle Database and another. For example, export files can archive database data or move data among different databases that run on the same or different operating systems.

2

Creating an Oracle Database

This chapter discusses the process of creating an Oracle Database, and contains the following topics:

- Deciding How to Create an Oracle Database
- Manually Creating an Oracle Database
- Understanding the CREATE DATABASE Statement
- Initialization Parameters and Database Creation
- Troubleshooting Database Creation
- Dropping a Database
- Managing Initialization Parameters Using a Server Parameter File
- Defining Application Services for Oracle Database 10g
- Considerations After Creating a Database
- Viewing Information About the Database

See Also:

- Part III, "Automated File and Storage Management", for information about creating a database whose underlying operating system files are automatically created and managed by the Oracle Database server
- *Oracle Real Application Clusters Installation and Configuration Guide* for additional information specific to an Oracle Real Application Clusters environment

Deciding How to Create an Oracle Database

You can create an Oracle Database in three ways:

- Use the Database Configuration Assistant (DBCA).

  DBCA can be launched by the Oracle Universal Installer, depending upon the type of install that you select, and provides a graphical user interface (GUI) that guides you through the creation of a database. You can also launch DBCA as a standalone tool at any time after Oracle Database installation to create or make a copy (clone) of a database. Please refer to *Oracle 2 Day DBA* for details information on creating a database using DBCA.

- Use the CREATE DATABASE statement.

  You can use the CREATE DATABASE SQL statement to create a database. If you do so, you must complete additional actions before you have an operational database. These actions include creating users and temporary tablespaces, building views of the data dictionary tables, and installing Oracle built-in packages. These actions can be performed by executing prepared scripts, many of which are supplied for you.

  If you have existing scripts for creating your database, consider editing those scripts to take advantage of new Oracle Database features. Oracle provides a sample database creation script and a sample initialization parameter file with the Oracle Database software files. Both the script and the file can be edited to suit your needs. See "Manually Creating an Oracle Database" on page 2-2.

- Upgrade an existing database.

  If you are already using a earlier release of Oracle Database, database creation is required only if you want an entirely new database. You can upgrade your existing Oracle Database and use it with the new release of the database software. The *Oracle Database Upgrade Guide* manual contains information about upgrading an existing Oracle Database.

The remainder of this chapter discusses creating a database manually.

Manually Creating an Oracle Database

This section takes you through the planning stage and the actual creation of the database.

Considerations Before Creating the Database

Database creation prepares several operating system files to work together as an Oracle Database. You need only create a database once, regardless of how many datafiles it has or how many instances access it. You can create a database to erase information in an existing database and create a new database with the same name and physical structure.

The following topics can help prepare you for database creation.

- Planning for Database Creation
- Meeting Creation Prerequisites

Planning for Database Creation

Prepare to create the database by research and careful planning. Table 2–1 lists some recommended actions:

Table 2–1 Planning for Database Creation

Action	Additional Information
Plan the database tables and indexes and estimate the amount of space they will require.	Part II, "Oracle Database Structure and Storage" Part IV, "Schema Objects"
Plan the layout of the underlying operating system files your database will comprise. Proper distribution of files can improve database performance dramatically by distributing the I/O during file access. You can distribute I/O in several ways when you install Oracle software and create your database. For example, you can place redo log files on separate disks or use striping. You can situate datafiles to reduce contention. And you can control data density (number of rows to a data block).	*Oracle Database Performance Tuning Guide* Your Oracle operating system specific documentation
Consider using Oracle-managed files and Automatic Storage Management to create and manage the operating system files that make up your database storage.	Part III, "Automated File and Storage Management"
Select the **global database name**, which is the name and location of the database within the network structure. Create the global database name by setting both the `DB_NAME` and `DB_DOMAIN` initialization parameters.	"Determining the Global Database Name" on page 2-28

Table 2–1 (Cont.) Planning for Database Creation

Action	Additional Information
Familiarize yourself with the initialization parameters will be contained in the initialization parameter file. Become familiar with the concept and operation of a **server parameter file**. A server parameter file lets you store and manage your initialization parameters persistently in a server-side disk file.	"Initialization Parameters and Database Creation" on page 2-27 "What Is a Server Parameter File?" on page 2-46 *Oracle Database Reference*
Select the database character set. All character data, including data in the data dictionary, is stored in the database character set. You must specify the database character set when you create the database. If clients using different character sets will access the database, then choose a superset that includes all client character sets. Otherwise, character conversions may be necessary at the cost of increased overhead and potential data loss. You can also specify an alternate character set.	*Oracle Database Globalization Support Guide*
Consider what time zones your database must support. Oracle Database uses one of two time zone files, located in the Oracle home directory, as the source of valid time zones. If you need to use a time zone that is not in the default time zone file (`timezone.dat`), but that is present in the larger time zone file (`timezlrg.dat`), then you must set the `ORA_TZFILE` environment variable to point to the larger file.	"Specifying the Database Time Zone File" on page 2-25
Select the standard database block size. This is specified at database creation by the `DB_BLOCK_SIZE` initialization parameter and cannot be changed after the database is created. The `SYSTEM` tablespace and most other tablespaces use the standard block size. Additionally, you can specify up to four nonstandard block sizes when creating tablespaces.	"Specifying Database Block Sizes" on page 2-31
Determine the appropriate initial sizing for the `SYSAUX` tablespace.	"Creating the SYSAUX Tablespace" on page 2-17
Plan to use a default tablespace for non-`SYSTEM` users to prevent inadvertent saving of database objects in the `SYSTEM` tablespace.	"Creating a Default Permanent Tablespace" on page 2-20
Plan to use an undo tablespace, rather than rollback segments, to manage your undo data.	Chapter 10, "Managing the Undo Tablespace"

Table 2–1 (Cont.) Planning for Database Creation

Action	Additional Information
Develop a backup and recovery strategy to protect the database from failure. It is important to protect the control file by multiplexing, to choose the appropriate backup mode, and to manage the online and archived redo logs.	Chapter 6, "Managing the Redo Log" Chapter 7, "Managing Archived Redo Logs" Chapter 5, "Managing Control Files" *Oracle Database Backup and Recovery Basics*
Familiarize yourself with the principles and options of starting up and shutting down an instance and mounting and opening a database.	Chapter 3, "Starting Up and Shutting Down"

Meeting Creation Prerequisites

Before you can create a new database, the following prerequisites must be met:

- The desired Oracle software must be installed. This includes setting various environment variables unique to your operating system and establishing the directory structure for software and database files.
- You must have the operating system privileges associated with a fully operational database administrator. You must be specially authenticated by your operating system or through a password file, allowing you to start up and shut down an instance before the database is created or opened. This authentication is discussed in "Database Administrator Authentication" on page 1-12.
- Sufficient memory must be available to start the Oracle Database instance.
- Sufficient disk storage space must be available for the planned database on the computer that runs Oracle Database.

All of these are discussed in the Oracle Database installation guide specific to your operating system. If you use the Oracle Universal Installer, it will guide you through your installation and provide help in setting environment variables and establishing directory structure and authorizations.

Creating the Database

This section presents the steps involved when you create a database manually. These steps should be followed in the order presented. The prerequisites described

in the preceding section must already have been completed. That is, you have established the environment for creating your Oracle Database, including most operating system dependent environmental variables, as part of the Oracle software installation process.

Step 1: Decide on Your Instance Identifier (SID)

Step 2: Establish the Database Administrator Authentication Method

Step 3: Create the Initialization Parameter File

Step 4: Connect to the Instance

Step 5: Create a Server Parameter File (Recommended)

Step 6: Start the Instance

Step 7: Issue the CREATE DATABASE Statement

Step 8: Create Additional Tablespaces

Step 9: Run Scripts to Build Data Dictionary Views

Step 10: Run Scripts to Install Additional Options (Optional)

Step 11: Back Up the Database.

The examples shown in these steps create an example database `mynewdb`.

Notes:

- The steps in this section contain cross-references to other parts of this book and to other books. These cross-references take you to material that will help you to learn about and understand the initialization parameters and database structures with which you are not yet familiar.
- If you are using Oracle Automatic Storage Management to manage your disk storage, you must start the ASM instance and configure your disk groups before performing the following steps. For information about Automatic Storage Management, see Chapter 12, "Using Automatic Storage Management".

Step 1: Decide on Your Instance Identifier (SID)

An instance is made up of the system global area (SGA) and the background processes of an Oracle Database. Decide on a unique Oracle system identifier (SID)

for your instance and set the ORACLE_SID environment variable accordingly. This identifier is used to distinguish this instance from other Oracle Database instances that you may create later and run concurrently on your system.

The following example sets the SID for the instance and database we are about to create:

```
% setenv ORACLE_SID mynewdb
```

The value of the DB_NAME initialization parameter should match the SID setting.

Step 2: Establish the Database Administrator Authentication Method

You must be authenticated and granted appropriate system privileges in order to create a database. You can use the password file or operating system authentication method. Database administrator authentication and authorization is discussed in the following sections of this book:

- "Database Administrator Security and Privileges" on page 1-10
- "Database Administrator Authentication" on page 1-12
- "Creating and Maintaining a Password File" on page 1-20

Step 3: Create the Initialization Parameter File

The instance for any Oracle Database is started using an initialization parameter file. One way to create the initialization parameter file is to edit a copy of the sample initialization parameter file that Oracle provides on the distribution media, or the sample presented in this book.

For convenience, store your initialization parameter file in the Oracle Database default location, using the default name. Then when you start your database, it will not be necessary to specify the PFILE parameter of the STARTUP command, because Oracle Database automatically looks in the default location for the initialization parameter file.

Default initialization parameter file locations are shown in the following table:

Platform	Default Name	Default Location
UNIX	init*$ORACLE_HOME*.ora For example, the initialization parameter file for the mynewdb database is named: initmynewdb.ora	$ORACLE_HOME/dbs For example, the initialization parameter file for the mynewdb database is stored in the following location: /u01/oracle/dbs/initmynewdb.ora
Windows	init.ora	*ORACLE_BASE*\admin*SID*\pfile\init.ora

The following is the initialization parameter file used to create the mynewdb database on a UNIX system.

Sample Initialization Parameter File

```
control_files             = (/u0d/lcg03/control.001.dbf,
                             /u0d/lcg03/control.002.dbf,
                             /u0d/lcg03/control.003.dbf)
db_name                   = lcg03

db_domain                 = us.oracle.com

log_archive_dest_1        =
"LOCATION=/net/fstlcg03/private/yaliu/testlog/log.lcg03.fstlcg03/lcg03/arch"
log_archive_dest_state_1  = enable

db_block_size             = 8192
pga_aggregate_target      = 2500M

processes                 = 1000
sessions                  = 1200
open_cursors              = 1024

undo_management           = AUTO

shared_servers            = 3
remote_listener           = tnsfstlcg03

undo_tablespace           = smu_nd1
compatible                = 10.1.0.0.0

sga_target                = 1500M
```

```
nls_language               = AMERICAN
nls_territory              = AMERICA
db_recovery_file_dest      =
/net/fstlcg03/private/yaliu/testlog/log.lcg03.fstlcg03/lcg03/arch
db_recovery_file_dest_size = 100G
```

See Also:

- "Initialization Parameters and Database Creation" on page 2-27 for more information on some of these parameters and other initialization parameters that you can include

Step 4: Connect to the Instance

Start SQL*Plus and connect to your Oracle Database instance AS SYSDBA.

```
$ SQLPLUS /nolog
CONNECT SYS/password AS SYSDBA
```

Step 5: Create a Server Parameter File (Recommended)

Oracle recommends that you create a server parameter file as a dynamic means of maintaining initialization parameters. The server parameter file is discussed in "Managing Initialization Parameters Using a Server Parameter File" on page 2-45.

The following script creates a server parameter file from the text initialization parameter file and writes it to the default location. The script can be executed before or after instance startup, but after you connect as SYSDBA. The database must be restarted before the server parameter file takes effect.

```
-- create the server parameter file
CREATE SPFILE='/u01/oracle/dbs/spfilemynewdb.ora' FROM
       PFILE='/u01/oracle/admin/initmynewdb/scripts/init.ora';
SHUTDOWN
-- the next startup will use the server parameter file
EXIT
```

Step 6: Start the Instance

Start an instance without mounting a database. Typically, you do this only during database creation or while performing maintenance on the database. Use the STARTUP command with the NOMOUNT clause. In this example, because the server parameter file is stored in the default location, you are not required to specify the PFILE clause:

```
STARTUP NOMOUNT
```

At this point, the SGA is created and background processes are started in preparation for the creation of a new database. The database itself does not yet exist.

See Also:

- "Managing Initialization Parameters Using a Server Parameter File" on page 2-45
- Chapter 3, "Starting Up and Shutting Down", to learn how to use the STARTUP command

Step 7: Issue the CREATE DATABASE Statement

To create the new database, use the CREATE DATABASE statement. The following statement creates database mynewdb:

```
CREATE DATABASE mynewdb
   USER SYS IDENTIFIED BY pz6r58
   USER SYSTEM IDENTIFIED BY y1tz5p
   LOGFILE GROUP 1 ('/u01/oracle/oradata/mynewdb/redo01.log') SIZE 100M,
           GROUP 2 ('/u01/oracle/oradata/mynewdb/redo02.log') SIZE 100M,
           GROUP 3 ('/u01/oracle/oradata/mynewdb/redo03.log') SIZE 100M
   MAXLOGFILES 5
   MAXLOGMEMBERS 5
   MAXLOGHISTORY 1
   MAXDATAFILES 100
   MAXINSTANCES 1
   CHARACTER SET US7ASCII
   NATIONAL CHARACTER SET AL16UTF16
   DATAFILE '/u01/oracle/oradata/mynewdb/system01.dbf' SIZE 325M REUSE
   EXTENT MANAGEMENT LOCAL
   SYSAUX DATAFILE '/u01/oracle/oradata/mynewdb/sysaux01.dbf' SIZE 325M REUSE
   DEFAULT TABLESPACE tbs_1
   DEFAULT TEMPORARY TABLESPACE tempts1
      TEMPFILE '/u01/oracle/oradata/mynewdb/temp01.dbf'
      SIZE 20M REUSE
   UNDO TABLESPACE undotbs
      DATAFILE '/u01/oracle/oradata/mynewdb/undotbs01.dbf'
      SIZE 200M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;
```

A database is created with the following characteristics:

- The database is named `mynewdb`. Its global database name is `mynewdb.us.oracle.com`. See "DB_NAME Initialization Parameter" and "DB_DOMAIN Initialization Parameter" on page 2-29.
- Three control files are created as specified by the `CONTROL_FILES` initialization parameter, which was set before database creation in the initialization parameter file. See "Sample Initialization Parameter File" on page 2-8 and "Specifying Control Files" on page 2-30.
- The password for user `SYS` is `pz6r58` and the password for `SYSTEM` is `y1tz5p`. The two clauses that specify the passwords for `SYS` and `SYSTEM` are not mandatory in this release of Oracle Database. However, if you specify either clause, you must specify both clauses. For further information about the use of these clauses, see "Protecting Your Database: Specifying Passwords for Users SYS and SYSTEM" on page 2-15.
- The new database has three redo log files as specified in the `LOGFILE` clause. `MAXLOGFILES`, `MAXLOGMEMBERS`, and `MAXLOGHISTORY` define limits for the redo log. See Chapter 6, "Managing the Redo Log".
- `MAXDATAFILES` specifies the maximum number of datafiles that can be open in the database. This number affects the initial sizing of the control file.

  Note: You can set several limits during database creation. Some of these limits are limited by and affected by operating system limits. For example, if you set `MAXDATAFILES`, Oracle Database allocates enough space in the control file to store `MAXDATAFILES` filenames, even if the database has only one datafile initially. However, because the maximum control file size is limited and operating system dependent, you might not be able to set all `CREATE DATABASE` parameters at their theoretical maximums.

  For more information about setting limits during database creation, see the *Oracle Database SQL Reference* and your operating system specific Oracle documentation.

- `MAXINSTANCES` specifies that only one instance can have this database mounted and open.
- The `US7ASCII` character set is used to store data in this database.

- The AL16UTF16 character set is specified as the NATIONAL CHARACTER SET, used to store data in columns specifically defined as NCHAR, NCLOB, or NVARCHAR2.
- The SYSTEM tablespace, consisting of the operating system file /u01/oracle/oradata/mynewdb/system01.dbf is created as specified by the DATAFILE clause. If a file with that name already exists, it is overwritten.
- The SYSTEM tablespace is a locally managed tablespace. See "Creating a Locally Managed SYSTEM Tablespace" on page 2-15.
- A SYSAUX tablespace is created, consisting of the operating system file /u01/oracle/oradata/mynewdb/sysaux01.dbf as specified in the SYSAUX DATAFILE clause. See "Creating the SYSAUX Tablespace" on page 2-17.
- The DEFAULT TABLESPACE clause creates and names a default permanent tablespace for this database.
- The DEFAULT TEMPORARY TABLESPACE clause creates and names a default temporary tablespace for this database. See "Creating a Default Temporary Tablespace" on page 2-20.
- The UNDO TABLESPACE clause creates and names an undo tablespace that is used to store undo data for this database if you have specified UNDO_MANAGEMENT=AUTO in the initialization parameter file. See "Using Automatic Undo Management: Creating an Undo Tablespace" on page 2-19.
- Redo log files will not initially be archived, because the ARCHIVELOG clause is not specified in this CREATE DATABASE statement. This is customary during database creation. You can later use an ALTER DATABASE statement to switch to ARCHIVELOG mode. The initialization parameters in the initialization parameter file for mynewdb relating to archiving are LOG_ARCHIVE_DEST_1 and LOG_ARCHIVE_FORMAT. See Chapter 7, "Managing Archived Redo Logs".

See Also:

- "Understanding the CREATE DATABASE Statement" on page 2-14
- *Oracle Database SQL Reference* for more information about specifying the clauses and parameter values for the CREATE DATABASE statement

Step 8: Create Additional Tablespaces

To make the database functional, you need to create additional files and tablespaces for users. The following sample script creates some additional tablespaces:

```
CONNECT SYS/password AS SYSDBA
-- create a user tablespace to be assigned as the default tablespace for users
CREATE TABLESPACE users LOGGING
     DATAFILE '/u01/oracle/oradata/mynewdb/users01.dbf'
     SIZE 25M REUSE AUTOEXTEND ON NEXT  1280K MAXSIZE UNLIMITED
     EXTENT MANAGEMENT LOCAL;
-- create a tablespace for indexes, separate from user tablespace
CREATE TABLESPACE indx LOGGING
     DATAFILE '/u01/oracle/oradata/mynewdb/indx01.dbf'
     SIZE 25M REUSE AUTOEXTEND ON NEXT  1280K MAXSIZE UNLIMITED
     EXTENT MANAGEMENT LOCAL;
```

For information about creating tablespaces, see Chapter 8, "Managing Tablespaces".

Step 9: Run Scripts to Build Data Dictionary Views

Run the scripts necessary to build views, synonyms, and PL/SQL packages:

```
CONNECT SYS/password AS SYSDBA
@/u01/oracle/rdbms/admin/catalog.sql
@/u01/oracle/rdbms/admin/catproc.sql
EXIT
```

The following table contains descriptions of the scripts:

Script	Description
CATALOG.SQL	Creates the views of the data dictionary tables, the dynamic performance views, and public synonyms for many of the views. Grants PUBLIC access to the synonyms.
CATPROC.SQL	Runs all scripts required for or used with PL/SQL.

Step 10: Run Scripts to Install Additional Options (Optional)

You may want to run other scripts. The scripts that you run are determined by the features and options you choose to use or install. Many of the scripts available to you are described in the *Oracle Database Reference*.

If you plan to install other Oracle products to work with this database, see the installation instructions for those products. Some products require you to create

additional data dictionary tables. Usually, command files are provided to create and load these tables into the database data dictionary.

See your Oracle documentation for the specific products that you plan to install for installation and administration instructions.

Step 11: Back Up the Database.

Take a full backup of the database to ensure that you have a complete set of files from which to recover if a media failure occurs. For information on backing up a database, see *Oracle Database Backup and Recovery Basics.*

Understanding the CREATE DATABASE Statement

When you execute a CREATE DATABASE statement, Oracle Database performs (at least) a number of operations. The actual operations performed depend on the clauses that you specify in the CREATE DATABASE statement and the initialization parameters that you have set. Oracle Database performs at least these operations:

- Creates the datafiles for the database
- Creates the control files for the database
- Creates the redo log files for the database and establishes the ARCHIVELOG mode.
- Creates the SYSTEM tablespace and the SYSTEM rollback segment
- Creates the SYSAUX tablespace
- Creates the data dictionary
- Sets the character set that stores data in the database
- Sets the database time zone
- Mounts and opens the database for use

This section discusses several of the clauses of the CREATE DATABASE statement. It expands upon some of the clauses discussed in "Step 7: Issue the CREATE DATABASE Statement" on page 2-10 and introduces additional ones. Many of the CREATE DATABASES clauses discussed here can be used to simplify the creation and management of your database.

The following topics are contained in this section:

- Protecting Your Database: Specifying Passwords for Users SYS and SYSTEM

- Creating a Locally Managed SYSTEM Tablespace
- Creating the SYSAUX Tablespace
- Using Automatic Undo Management: Creating an Undo Tablespace
- Creating a Default Temporary Tablespace
- Specifying Oracle-Managed Files at Database Creation
- Supporting Bigfile Tablespaces During Database Creation
- Specifying the Database Time Zone and Time Zone File
- Specifying FORCE LOGGING Mode

Protecting Your Database: Specifying Passwords for Users SYS and SYSTEM

The clauses of the CREATE DATABASE statement used for specifying the passwords for users SYS and SYSTEM are:

- USER SYS IDENTIFIED BY *password*
- USER SYSTEM IDENTIFIED BY *password*

If you omit these clauses, these users are assigned the default passwords change_on_install and manager, respectively. A record is written to the alert file indicating that the default passwords were used. To protect your database, you should change these passwords using the ALTER USER statement immediately after database creation.

Oracle strongly recommends that you specify these clauses, even though they are optional in this release of Oracle Database. The default passwords are commonly known, and if you neglect to change them later, you leave database vulnerable to attack by malicious users.

See Also: "Some Security Considerations" on page 2-57

Creating a Locally Managed SYSTEM Tablespace

When you specify the EXTENT MANAGEMENT LOCAL clause in the CREATE DATABASE statement, you cause Oracle Database to create a locally managed SYSTEM tablespace, which is a tablespace for which the database determines and manages extent sizes. The COMPATIBLE initialization parameter must be set to 9.2 or higher for this statement to be successful. If you do not specify the EXTENT MANAGEMENT LOCAL clause, by default the database creates a dictionary-managed SYSTEM tablespace.

Locally managed tablespaces provide better performance and greater ease of management than dictionary-managed tablespaces. A locally managed SYSTEM tablespace is with AUTOALLOCATE enabled by default, which means that the system determines and controls the number and size of extents. You may notice an increase in the initial size of objects created in a locally managed SYSTEM tablespace because of the autoallocate policy. It is not possible to create a locally managed SYSTEM tablespace and specify UNIFORM extent size.

When you create your database with a locally managed SYSTEM tablespace, ensure the following conditions are met:

- A default temporary tablespace must exist, and that tablespace cannot be the SYSTEM tablespace.

  To meet this condition, you can specify the DEFAULT TEMPORARY TABLESPACE clause in the CREATE DATABASE statement, or you can omit the clause and let Oracle Database create the tablespace for you using a default name and in a default location.

- You must not create rollback segments in dictionary-managed tablespaces. Attempts to create rollback segments in a dictionary-managed tablespace will fail if the SYSTEM tablespace is locally managed.

  You can meet this condition by using automatic undo management. Oracle has deprecated the use of rollback segments. You can include the UNDO TABLESPACE clause in the CREATE DATABASE statement to create a specific undo tablespace. If you omit that clause, Oracle Database creates a locally managed undo tablespace for you using the default name and in a default location.

When the SYSTEM tablespace is locally managed, the following additional restrictions apply to other tablespaces in the database:

- You cannot create any dictionary-managed tablespaces in the database.
- You cannot migrate a locally managed tablespace to a dictionary-managed tablespace.
- Preexisting dictionary-managed tablespaces can remain in the database, but only in read-only mode. They cannot be altered to read / write.
- You can transport dictionary-managed tablespaces into the database, but you cannot set them to read / write.

Oracle Database lets you migrate an existing dictionary-managed SYSTEM tablespace to a locally managed tablespace using the DBMS_SPACE_ADMIN package. However, there is no procedure for backward migration.

See Also:

- *Oracle Database SQL Reference* for more specific information about the use of the DEFAULT TEMPORARY TABLESPACE and UNDO TABLESPACE clauses when EXTENT MANAGEMENT LOCAL is specified for the SYSTEM tablespace
- "Locally Managed Tablespaces" on page 8-4
- "Migrating the SYSTEM Tablespace to a Locally Managed Tablespace" on page 8-40

Creating the SYSAUX Tablespace

The SYSAUX tablespace is always created at database creation. The SYSAUX tablespace serves as an auxiliary tablespace to the SYSTEM tablespace. Because it is the default tablespace for many Oracle Database features and products that previously required their own tablespaces, it reduces the number of tablespaces required by the database and that you, as a DBA, must maintain. Other functionality or features that previously used the SYSTEM tablespace can now use the SYSAUX tablespace, thus reducing the load on the SYSTEM tablespace.

You can specify only datafile attributes for the SYSAUX tablespace, using the SYSAUX DATAFILE clause in the CREATE DATABASE statement. Mandatory attributes of the SYSAUX tablespace are set by Oracle Database and include:

- PERMANENT
- READ WRITE
- EXTENT MANAGMENT LOCAL
- SEGMENT SPACE MANAGMENT AUTO

You cannot alter these attributes with an ALTER TABLESPACE statement, and any attempt to do so will result in an error. You cannot drop or rename the SYSAUX tablespace.

The size of the SYSAUX tablespace is determined by the size of the database components that occupy SYSAUX. See Table 2–2 for a list of all SYSAUX occupants. Based on the initial sizes of these components, the SYSAUX tablespace needs to be at least 240 MB at the time of database creation. The space requirements of the SYSAUX tablespace will increase after the database is fully deployed, depending on the nature of its use and workload. For more information on how to manage the space consumption of the SYSAUX tablespace on an ongoing basis, please refer to the "Managing the SYSAUX Tablespace" on page 8-34.

If you include a DATAFILE clause for the SYSTEM tablespace, then you must specify the SYSAUX DATAFILE clause as well, or the CREATE DATABASE statement will fail. This requirement does not exist if the Oracle-managed files feature is enabled (see "Specifying Oracle-Managed Files at Database Creation" on page 2-21).

If you issue the CREATE DATABASE statement with no other clauses, then the software creates a default database with datafiles for the SYSTEM and SYSAUX tablespaces stored in system-determined default locations, or where specified by an Oracle-managed files initialization parameter.

The SYSAUX tablespace has the same security attributes as the SYSTEM tablespace.

> **Note:** This book discusses the creation of the SYSAUX database at database creation. When upgrading from a release of Oracle Database that did not require the SYSAUX tablespace, you must create the SYSAUX tablespace as part of the upgrade process. This is discussed in *Oracle Database Upgrade Guide*.

Table 2–2 lists the components that use the SYSAUX tablespace as their default tablespace during installation, and the tablespace in which they were stored in earlier releases:

Table 2–2 Database Components and the SYSAUX Tablespace

Component Using SYSAUX	Tablespace in Earlier Releases
Analytical Workspace Object Table	SYSTEM
Enterprise Manager Repository	OEM_REPOSITORY
LogMiner	SYSTEM
Logical Standby	SYSTEM
OLAP API History Tables	CWMLITE
Oracle Data Mining	ODM
Oracle Spatial	SYSTEM
Oracle Streams	SYSTEM
Oracle Text	DRSYS
Oracle Ultra Search	DRSYS
Oracle *inter*Media ORDPLUGINS Components	SYSTEM

Table 2–2 (Cont.) Database Components and the SYSAUX Tablespace

Component Using SYSAUX	Tablespace in Earlier Releases
Oracle *inter*Media ORDSYS Components	SYSTEM
Oracle *inter*Media SI_INFORMTN_SCHEMA Components	SYSTEM
Server Manageability Components	New in Oracle Database 10*g*
Statspack Repository	User-defined
Unified Job Scheduler	New in Oracle Database 10*g*
Workspace Manager	SYSTEM

The installation procedures for these components provide the means of establishing their occupancy of the SYSAUX tablespace.

> **See Also:** "Managing the SYSAUX Tablespace" on page 8-34 for information about managing the SYSAUX tablespace

Using Automatic Undo Management: Creating an Undo Tablespace

The use of rollback segments for storing rollback information has been deprecated. Instead, you should use automatic undo management, which in turn uses an undo tablespace. Doing so requires the use of a different set of initialization parameters, and optionally, the inclusion of the UNDO TABLESPACE clause in your CREATE DATABASE statement.

To enable automatic undo management mode, set the UNDO_MANAGEMENT initialization parameter to AUTO in your initialization parameter file. In this mode, rollback information, referred to as **undo data**, is stored in an undo tablespace rather than rollback segments and is managed by Oracle Database. If you want to define and name the undo tablespace yourself, you must also include the UNDO TABLESPACE clause at database creation time. If you omit this clause, and automatic undo management is enabled, then the database creates a default undo tablespace named SYS_UNDOTBS.

> **See Also:**
>
> - "Specifying the Method of Undo Space Management" on page 2-42
> - Chapter 10, "Managing the Undo Tablespace", for information about the creation and use of undo tablespaces

Creating a Default Permanent Tablespace

The DEFAULT TABLESPACE clause of the CREATE DATABASE statement specifies a default permanent tablespace for the database. Oracle Database assign to this tablespace any non-SYSTEM users for whom you do not explicitly specify a different permanent tablespace. If you do not specify this clause, then the SYSTEM tablespace is the default permanent tablespace for non-SYSTEM users. Oracle strongly recommends that you create a default permanent tablespace.

> **See Also:** *Oracle Database SQL Reference* for the syntax of the DEFAULT TABLESPACE clause of CREATE DATABASE and ALTER DATABASE

Creating a Default Temporary Tablespace

The DEFAULT TEMPORARY TABLESPACE clause of the CREATE DATABASE statement creates a default temporary tablespace for the database. Oracle Database this tablespace as the temporary tablespace for users who are not explicitly assigned a temporary tablespace.

You can explicitly assign a temporary tablespace or tablespace group to a user in the CREATE USER statement. However, if you do not do so, and if no default temporary tablespace has been specified for the database, then by default these users are assigned the SYSTEM tablespace as their temporary tablespace. It is not good practice to store temporary data in the SYSTEM tablespace, and it is cumbersome to assign every user a temporary tablespace individually. Therefore, Oracle recommends that you use the DEFAULT TEMPORARY TABLESPACE clause of CREATE DATABASE.

Note: When you specify a locally managed SYSTEM tablespace, the SYSTEM tablespace *cannot* be used as a temporary tablespace. In this case the database creates a default temporary tablespace. This behavior is explained in "Creating a Locally Managed SYSTEM Tablespace" on page 2-15.

You can add or the default temporary tablespace after database creation. You do this by creating a new temporary tablespace or tablespace group with a CREATE TEMPORARY TABLESPACE statement, and then assign it as the temporary tablespace using the ALTER DATABASE DEFAULT TEMPORARY TABLESPACE statement. Users will automatically be switched (or assigned) to the new default temporary tablespace.

The following statement assigns a new default temporary tablespace:

```
ALTER DATABASE DEFAULT TEMPORARY TABLESPACE tempts2;
```

The new default temporary tablespace must already exist. When using a locally managed SYSTEM tablespace, the new default temporary tablespace must also be locally managed.

You cannot drop or take offline a default temporary tablespace, but you can assign a new default temporary tablespace and then drop or take offline the former one. You cannot change a default temporary tablespace to a permanent tablespace.

Users can obtain the name of the current default temporary tablespace by querying the PROPERTY_NAME and PROPERTY_VALUE columns of the DATABASE_PROPERTIES view. These columns contain the values "DEFAULT_TEMP_TABLESPACE" and the default temporary tablespace name, respectively.

See Also:

- *Oracle Database SQL Reference* for the syntax of the DEFAULT TEMPORARY TABLESPACE clause of CREATE DATABASE and ALTER DATABASE
- "Temporary Tablespaces" on page 8-17 for information about creating and using temporary tablespaces
- "Multiple Temporary Tablespaces: Using Tablespace Groups" on page 8-21 for information about creating and using temporary tablespace groups

Specifying Oracle-Managed Files at Database Creation

You can minimize the number of clauses and parameters that you specify in your CREATE DATABASE statement by using the Oracle-managed files feature. You do this either by specifying a directory in which your files are created and managed by Oracle Database, or by using Automatic Storage Management. When you use Automatic Storage Management, you specify a disk group in which the database creates and manages your files, including file redundancy and striping.

By including any of the initialization parameters DB_CREATE_FILE_DEST, DB_CREATE_ONLINE_LOG_DEST_*n*, or DB_RECOVERY_FILE_DEST in your initialization parameter file, you instruct Oracle Database to create and manage the underlying operating system files of your database. Oracle Database will automatically create and manage the operating system files for the following

database structures, depending on which initialization parameters you specify and how you specify clauses in your CREATE DATABASE statement:

- Tablespaces
- Temporary tablespaces
- Control files
- Redo log files
- Archive log files
- Flashback logs
- Block change tracking files
- RMAN backups

> **See Also:** "Specifying a Flash Recovery Area" on page 2-29 for information about setting initialization parameters that create a flash recovery area

The following CREATE DATABASE statement shows briefly how the Oracle-managed files feature works, assuming you have specified required initialization parameters:

```
CREATE DATABASE rbdb1
     USER SYS IDENTIFIED BY pz6r58
     USER SYSTEM IDENTIFIED BY y1tz5p
     UNDO TABLESPACE undotbs
     DEFAULT TEMPORARY TABLESPACE tempts1;
```

- No DATAFILE clause is specified, so the database creates an Oracle-managed datafile for the SYSTEM tablespace.
- No LOGFILE clauses are included, so the database creates two Oracle-managed redo log file groups.
- No SYSAUX DATAFILE is included, so the database creates an Oracle-managed datafile for the SYSAUX tablespace.
- No DATAFILE subclause is specified for the UNDO TABLESPACE clause, so the database creates an Oracle-managed datafile for the undo tablespace.
- No TEMPFILE subclause is specified for the DEFAULT TEMPORARY TABLESPACE clause, so the database creates an Oracle-managed tempfile.

- If no CONTROL_FILES initialization parameter is specified in the initialization parameter file, then the database also creates an Oracle-managed control file.
- If you are using a server parameter file (see "Managing Initialization Parameters Using a Server Parameter File" on page 2-45), the database automatically sets the appropriate initialization parameters.

See Also:

- Chapter 11, "Using Oracle-Managed Files", for information about the Oracle-managed files feature and how to use it
- Chapter 12, "Using Automatic Storage Management", for information about Automatic Storage Management

Supporting Bigfile Tablespaces During Database Creation

Oracle Database simplifies management of tablespaces and enables support for ultra-large databases by letting you create **bigfile tablespaces**. Bigfile tablespaces can contain only one file, but that file can have up to 4G blocks. The maximum number of datafiles in an Oracle Database is limited (usually to 64K files). Therefore, bigfile tablespaces can significantly enhance the storage capacity of an Oracle Database.

This section discusses the clauses of the CREATE DATABASE statement that let you include support for bigfile tablespaces.

See Also: "Bigfile Tablespaces" on page 8-9 for more information about bigfile tablespaces

Specifying the Default Tablespace Type

The SET DEFAULT ... TABLESPACE clause of the CREATE DATABASE statement to determines the default type of tablespace for this database in subsequent CREATE TABLESPACE statements. Specify either SET DEFAULT BIGFILE TABLESPACE or SET DEFAULT SMALLFILE TABLESPACE. If you omit this clause, the default is a **smallfile tablespace**, which is the traditional type of Oracle Database tablespace. A smallfile tablespace can contain up to 1022 files with up to 4M blocks each.

The use of bigfile tablespaces further enhances the Oracle-managed files feature, because bigfile tablespaces make datafiles completely transparent for users. SQL syntax for the ALTER TABLESPACE statement has been extended to allow you to perform operations on tablespaces, rather than the underlying datafiles.

The CREATE DATABASE statement shown in "Specifying Oracle-Managed Files at Database Creation" on page 2-21 can be modified as follows to specify that the default type of tablespace is a bigfile tablespace:

```
CREATE DATABASE rbdb1
     USER SYS IDENTIFIED BY pz6r58
     USER SYSTEM IDENTIFIED BY y1tz5p
     SET DEFAULT BIGFILE TABLESPACE
     UNDO TABLESPACE undotbs
     DEFAULT TEMPORARY TABLESPACE tempts1;
```

To dynamically change the default tablespace type after database creation, use the SET DEFAULT TABLESPACE clause of the ALTER DATABASE statement:

```
ALTER DATABASE SET DEFAULT BIGFILE TABLESPACE;
```

You can determine the current default tablespace type for the database by querying the DATABASE_PROPERTIES data dictionary view as follows:

```
SELECT PROPERTY_VALUE FROM DATABASE_PROPERTIES
   WHERE PROPERTY_NAME = 'DEFAULT_TBS_TYPE';
```

Overriding the Default Tablespace Type

The SYSTEM and SYSAUX tablespaces are always created with the default tablespace type. However, you can explicitly override the default tablespace type for the UNDO and DEFAULT TEMPORARY tablespace during the CREATE DATABASE operation.

For example, you can create a bigfile UNDO tablespace in a database with the default tablespace type of smallfile as follows:

```
CREATE DATABASE rbdb1
...
     BIGFILE UNDO TABLESPACE undotbs
        DATAFILE '/u01/oracle/oradata/mynewdb/undotbs01.dbf'
        SIZE 200M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;
```

You can create a smallfile DEFAULT TEMPORARY tablespace in a database with the default tablespace type of bigfile as follows:

```
CREATE DATABASE rbdb1
   SET DEFAULT BIGFILE TABLSPACE
...
     SMALLFILE DEFAULT TEMPORARY TABLESPACE tempts1
        TEMPFILE '/u01/oracle/oradata/mynewdb/temp01.dbf'
        SIZE 20M REUSE
...
```

Specifying the Database Time Zone and Time Zone File

Oracle Database lets you specify the database default time zone and lets you choose the supporting time zone file.

Specifying the Database Time Zone

You set the database default time zone by specifying the `SET TIME_ZONE` clause of the `CREATE DATABASE` statement. If omitted, the default database time zone is the operating system time zone. The database time zone can be changed for a session with an `ALTER SESSION` statement.

> **See Also:** *Oracle Database Globalization Support Guide* for more information about setting the database time zone

Specifying the Database Time Zone File

This section provides information on the time zone files used to support the database time zone, specifically on Solaris platforms. Names of directories, filenames, and environment variables may differ for on other platforms but will probably be the same for all UNIX platforms.

Two time zone files are provided in the Oracle Database installation directory:

- `$ORACLE_HOME/oracore/zoneinfo/timezone.dat`

  This is the default file. It contains the most commonly used time zones and is smaller, thus enabling better database performance.

- `$ORACLE_HOME/oracore/zoneinfo/timezlrg.dat`

  This file contains a larger set of defined time zones and should be used if you require zones that are not defined in the default `timezone.dat` file. Use of this larger set of time zone information can affect performance.

To enable the use of the larger time zone datafile, do the following:

1. Shut down the database.
2. Set the environment variable `ORA_TZFILE` to the full path name of the location for the `timezlrg.dat` file.
3. Restart the database.

Once the larger `timezlrg.dat` is used, it must continue to be used unless you are sure that none of the nondefault zones are used for data that is stored in the database. Also, all databases that share information must use the same time zone datafile.

The time zone files contain the valid time zone names. The following information is included for each zone (note that abbreviations are only used in conjunction with the zone names):

- Offset from UTC (formerly GMT)
- Transition times for daylight savings
- Abbreviation for standard time
- Abbreviation for daylight savings time

To view the time zone names in the file being used by your database, use the following query:

```
SELECT * FROM V$TIMEZONE_NAMES;
```

The time zone names contained in both the larger and smaller time zone files are listed in the *Oracle Database Globalization Support Guide*.

Specifying FORCE LOGGING Mode

Some data definition language statements (such as CREATE TABLE) allow the `NOLOGGING` clause, which causes some database operations not to generate redo records in the database redo log. The `NOLOGGING` setting can speed up operations that can be easily recovered outside of the database recovery mechanisms, but it can negatively affect media recovery and standby databases.

Oracle Database lets you force the writing of redo records even when `NOLOGGING` has been specified in DDL statements. The database never generates redo records for temporary tablespaces and temporary segments, so forced logging has no affect for objects.

> **See Also:** *Oracle Database SQL Reference* for information about operations that can be done in `NOLOGGING` mode

Using the FORCE LOGGING Clause

To put the database into `FORCE LOGGING` mode, use the `FORCE LOGGING` clause in the `CREATE DATABASE` statement. If you do not specify this clause, the database is not placed into `FORCE LOGGING` mode.

Use the `ALTER DATABASE` statement to place the database into `FORCE LOGGING` mode after database creation. This statement can take a considerable time for completion, because it waits for all unlogged direct writes to complete.

You can cancel `FORCE LOGGING` mode using the following SQL statement:

```
ALTER DATABASE NO FORCE LOGGING;
```

Independent of specifying FORCE LOGGING for the database, you can selectively specify FORCE LOGGING or NO FORCE LOGGING at the tablespace level. However, if FORCE LOGGING mode is in effect for the database, it takes precedence over the tablespace setting. If it is not in effect for the database, then the individual tablespace settings are enforced. Oracle recommends that either the entire database is placed into FORCE LOGGING mode, or individual tablespaces be placed into FORCE LOGGING mode, but not both.

The FORCE LOGGING mode is a persistent attribute of the database. That is, if the database is shut down and restarted, it remains in the same logging mode. However, if you re-create the control file, the database is not restarted in the FORCE LOGGING mode unless you specify the FORCE LOGGING clause in the CREATE CONTROL FILE statement.

> **See Also:** "Controlling the Writing of Redo Records" on page 8-24 for information about using the FORCE LOGGING clause for tablespace creation.

Performance Considerations of FORCE LOGGING Mode

FORCE LOGGING mode results in some performance degradation. If the primary reason for specifying FORCE LOGGING is to ensure complete media recovery, and there is no standby database active, then consider the following:

- How many media failures are likely to happen?
- How serious is the damage if unlogged direct writes cannot be recovered?
- Is the performance degradation caused by forced logging tolerable?

If the database is running in NOARCHIVELOG mode, then generally there is no benefit to placing the database in FORCE LOGGING mode. Media recovery is not possible in NOARCHIVELOG mode, so if you combine it with FORCE LOGGING, the result may be performance degradation with little benefit.

Initialization Parameters and Database Creation

Oracle Database has provided generally appropriate values in the sample initialization parameter file provided with your database software or created for you by the Database Configuration Assistant. You can edit these Oracle-supplied initialization parameters and add others, depending upon your configuration and options and how you plan to tune the database. For any relevant initialization

parameters not specifically included in the initialization parameter file, the database supplies defaults.

If you are creating an Oracle Database for the first time, Oracle suggests that you minimize the number of parameter values that you alter. As you become more familiar with your database and environment, you can dynamically tune many initialization parameters using the `ALTER SYSTEM` statement. If you are using a traditional text initialization parameter file, your changes are effective only for the current instance. To make them permanent, you must update them manually in the initialization parameter file, or they will be lost over the next shutdown and startup of the database. If you are using a server parameter file, initialization parameter file changes made by the `ALTER SYSTEM` statement can persist across shutdown and startup. This is discussed in "Managing Initialization Parameters Using a Server Parameter File" on page 2-45.

This section introduces you to some of the basic initialization parameters you can add or edit before you create your new database.

The following topics are contained in this section:

- Determining the Global Database Name
- Specifying a Flash Recovery Area
- Specifying Control Files
- Specifying Database Block Sizes
- Managing the System Global Area (SGA)
- Specifying the Maximum Number of Processes
- Specifying the Method of Undo Space Management
- The COMPATIBLE Initialization Parameter and Irreversible Compatibility
- Setting the License Parameter

> **See Also:** *Oracle Database Reference* for descriptions of all initialization parameters including their default settings

Determining the Global Database Name

The global database name consists of the user-specified local database name and the location of the database within a network structure. The `DB_NAME` initialization parameter determines the local name component of the database name, and the `DB_DOMAIN` parameter indicates the domain (logical location) within a network

structure. The combination of the settings for these two parameters must form a database name that is unique within a network.

For example, to create a database with a global database name of `test.us.acme.com`, edit the parameters of the new parameter file as follows:

```
DB_NAME = test
DB_DOMAIN = us.acme.com
```

You can rename the `GLOBAL_NAME` of your database using the `ALTER DATABASE RENAME GLOBAL_NAME` statement. However, you must also shut down and restart the database after first changing the `DB_NAME` and `DB_DOMAIN` initialization parameters and re-creating the control file.

> **See Also:** *Oracle Database Utilities* for information about using the `DBNEWID` utility, which is another means of changing a database name

DB_NAME Initialization Parameter

`DB_NAME` must be set to a text string of no more than eight characters. During database creation, the name provided for `DB_NAME` is recorded in the datafiles, redo log files, and control file of the database. If during database instance startup the value of the `DB_NAME` parameter (in the parameter file) and the database name in the control file are not the same, the database does not start.

DB_DOMAIN Initialization Parameter

`DB_DOMAIN` is a text string that specifies the network domain where the database is created. This is typically the name of the organization that owns the database. If the database you are about to create will ever be part of a distributed database system, give special attention to this initialization parameter before database creation.

> **See Also:** Part VII, "Distributed Database Management" for more information about distributed databases

Specifying a Flash Recovery Area

A flash recovery area is a location in which Oracle Database can store and manage files related to backup and recovery. It is distinct from the database area, which is a location for the Oracle-managed current database files (datafiles, control files, and online redo logs).

You specify a flash recovery area with the following initialization parameters:

- DB_RECOVERY_FILE_DEST: Location of the flash recovery area. This can be a directory, file system, or Automatic Storage Management (ASM) disk group. It cannot be a raw file system.

  In a RAC environment, this location must be on a cluster file system, ASM disk group, or a shared directory configured through NFS.

- DB_RECOVERY_FILE_DEST_SIZE: Specifies the maximum total bytes to be used by the flash recovery area. This initialization parameter must be specified before DB_RECOVERY_FILE_DEST is enabled.

In a RAC environment, the settings for these two parameters must be the same on all instances.

You cannot enable these parameters if you have set values for the LOG_ARCHIVE_DEST and LOG_ARCHIVE_DUPLEX_DEST parameters. You must disable those parameters before setting up the flash recovery area. You can instead set values for the LOG_ARCHIVE_DEST_*n* parameters. If you do not set values for local LOG_ARCHIVE_DEST_*n*, then setting up the flash recovery area will implicitly set LOG_ARCHIVE_DEST_10 to the flash recovery area.

Oracle recommends using a flash recovery area, because it can simplify backup and recovery operations for your database.

> **See Also:** *Oracle Database Backup and Recovery Basics* to learn how to create and use a flash recovery area

Specifying Control Files

The CONTROL_FILES initialization parameter specifies one or more control filenames for the database. When you execute the CREATE DATABASE statement, the control files listed in the CONTROL_FILES parameter are created.

If do not include CONTROL_FILES in the initialization parameter file, then Oracle Database creates a control file using a default operating system dependent filename or, if you have enabled Oracle-managed files, creates Oracle-managed control files .

If you want the database to create new operating system files when creating database control files, the filenames listed in the CONTROL_FILES parameter must not match any filenames that currently exist on your system. If you want the database to reuse or overwrite existing files when creating database control files, ensure that the filenames listed in the CONTROL_FILES parameter match the filenames that are to be reused.

Caution: Use extreme caution when setting this specifying CONTROL_FILE filenames. If you inadvertently specify a file that already exists and execute the CREATE DATABASE statement, the previous contents of that file will be overwritten.

Oracle strongly recommends you use at least two control files stored on separate physical disk drives for each database.

See Also:

- Chapter 5, "Managing Control Files"
- "Specifying Oracle-Managed Files at Database Creation" on page 2-21

Specifying Database Block Sizes

The DB_BLOCK_SIZE initialization parameter specifies the standard block size for the database. This block size is used for the SYSTEM tablespace and by default in other tablespaces. Oracle Database can support up to four additional nonstandard block sizes.

DB_BLOCK_SIZE Initialization Parameter

The most commonly used block size should be picked as the standard block size. In many cases, this is the only block size that you need to specify. Typically, DB_BLOCK_SIZE is set to either 4K or 8K. If you do not set a value for this parameter, the default data block size is operating system specific, which is generally adequate.

You cannot change the block size after database creation except by re-creating the database. If the database block size is different from the operating system block size, ensure that the database block size is a multiple of the operating system block size. For example, if your operating system block size is 2K (2048 bytes), the following setting for the DB_BLOCK_SIZE initialization parameter is valid:

```
DB_BLOCK_SIZE=4096
```

A larger data block size provides greater efficiency in disk and memory I/O (access and storage of data). Therefore, consider specifying a block size larger than your operating system block size if the following conditions exist:

- Oracle Database is on a large computer system with a large amount of memory and fast disk drives. For example, databases controlled by mainframe

computers with vast hardware resources typically use a data block size of 4K or greater.

- The operating system that runs Oracle Database uses a small operating system block size. For example, if the operating system block size is 1K and the default data block size matches this, the database may be performing an excessive amount of disk I/O during normal operation. For best performance in this case, a database block should consist of multiple operating system blocks.

  See Also: Your operating system specific Oracle documentation for details about the default block size.

Nonstandard Block Sizes

Tablespaces of nonstandard block sizes can be created using the `CREATE TABLESPACE` statement and specifying the `BLOCKSIZE` clause. These nonstandard block sizes can have any of the following power-of-two values: 2K, 4K, 8K, 16K or 32K. Platform-specific restrictions regarding the maximum block size apply, so some of these sizes may not be allowed on some platforms.

To use nonstandard block sizes, you must configure subcaches within the buffer cache area of the SGA memory for all of the nonstandard block sizes that you intend to use. The initialization parameters used for configuring these subcaches are described in the next section, "Managing the System Global Area (SGA)".

The ability to specify multiple block sizes for your database is especially useful if you are transporting tablespaces between databases. You can, for example, transport a tablespace that uses a 4K block size from an OLTP environment to a data warehouse environment that uses a standard block size of 8K.

See Also:

- "Creating Tablespaces" on page 8-3
- "Transporting Tablespaces Between Databases" on page 8-40

Managing the System Global Area (SGA)

This section discusses the initialization parameters that affect the amount of memory allocated to the System Global Area (SGA). Except for the `SGA_MAX_SIZE` initialization parameter, they are dynamic parameters whose values can be changed by the `ALTER SYSTEM` statement. The size of the SGA is dynamic, and can grow or shrink by dynamically altering these parameters.

This section contains the following topics:

- Components and Granules in the SGA
- Limiting the Size of the SGA
- Using Automatic Shared Memory Management
- Setting the Buffer Cache Initialization Parameters
- Using Manual Shared Memory Management
- Viewing Information About the SGA

See Also:

- *Oracle Database Performance Tuning Guide* for information about tuning the components of the SGA
- *Oracle Database Concepts* for a conceptual discussion of automatic shared memory management

Components and Granules in the SGA

The SGA comprises a number of memory **components**, which are pools of memory used to satisfy a particular class of memory allocation requests. Examples of memory components include the shared pool (used to allocate memory for SQL and PL/SQL execution), the java pool (used for java objects and other java execution memory), and the buffer cache (used for caching disk blocks). All SGA components allocate and deallocate space in units of **granules**. Oracle Database tracks SGA memory use in internal numbers of granules for each SGA component.

The memory for dynamic components in the SGA is allocated in the unit of granules. Granule size is determined by total SGA size. Generally speaking, on most platforms, if the total SGA size is equal to or less than 1 GB, then granule size is 4 MB. For SGAs larger than 1 GB, granule size is 16 MB. Some platform dependencies may arise. For example, on 32-bit Windows NT, the granule size is 8 MB for SGAs larger than 1 GB. Consult your operating system specific documentation for more details.

You can query the `V$SGAINFO` view to see the granule size that is being used by an instance. The same granule size is used for all dynamic components in the SGA.

If you specify a size for a component that is not a multiple of granule size, Oracle Database rounds the specified size up to the nearest multiple. For example, if the granule size is 4 MB and you specify `DB_CACHE_SIZE` as 10 MB, the database actually allocates 12 MB.

Limiting the Size of the SGA

The SGA_MAX_SIZE initialization parameter specifies the maximum size of the System Global Area for the lifetime of the instance. You can dynamically alter the initialization parameters affecting the size of the buffer caches, shared pool, large pool, Java pool, and streams pool but only to the extent that the sum of these sizes and the sizes of the other components of the SGA (fixed SGA, variable SGA, and redo log buffers) does not exceed the value specified by SGA_MAX_SIZE.

If you do not specify SGA_MAX_SIZE, then Oracle Database selects a default value that is the sum of all components specified or defaulted at initialization time. If you do specify SGA_MAX_SIZE, and at the time the database is initialized the value is less than the sum of the memory allocated for all components, either explicitly in the parameter file or by default, then the database ignores the setting for SGA_MAX_SIZE.

Using Automatic Shared Memory Management

You enable the automatic shared memory management feature by setting the SGA_TARGET parameter. This parameter in effect replaces the parameters that control the memory allocated for a specific set of individual components, which are now automatically managed. In addition, you must ensure that the STATISTICS_LEVEL initialization parameter is set to TYPICAL (the default) or ALL.

The SGA_TARGET initialization parameter reflects the total size of the SGA. Table 2–3 lists the SGA components for which SGA_TARGET includes memory and the initialization parameters corresponding to those components.

Table 2–3 Automatically Sized SGA Components and Corresponding Parameters

SGA Component	Initialization Parameter
Fixed SGA and other internal allocations needed by the Oracle Database instance	NA
The shared pool	SHARED_POOL_SIZE
The large pool	LARGE_POOL_SIZE
The Java pool	JAVA_POOL_SIZE
The buffer cache	DB_CACHE_SIZE

The parameters listed in Table 2–4, if they are set, take their memory from SGA_TARGET, leaving what is available for the four components listed in Table 2–3.

Table 2–4 Manually Sized SGA Components that Use SGA_TARGET Space

SGA Component	Initialization Parameter
The log buffer	LOG_BUFFER
The keep and recycle buffer caches	DB_KEEP_CACHE_SIZE DB_RECYCLE_CACHE_SIZE
Nonstandard block size buffer caches	DB_nK_CACHE_SIZE
The Streams pool	STREAMS_POOL_SIZE

When you are migrating from a manual management scheme, execute the following query on the instance running in manual mode to get a value for SGA_TARGET:

```
SELECT (
   (SELECT SUM(value) FROM V$SGA) -
   (SELECT CURRENT_SIZE FROM V$SGA_DYNAMIC_FREE_MEMORY)
   ) "SGA_TARGET"
FROM DUAL;
```

You can then remove the parameters that formerly controlled memory for the now automatically sized components.

For example, suppose you currently have the following configuration of parameters on a manual mode instance with SGA_MAX_SIZE set to 1200M:

- SHARED_POOL_SIZE = 200M
- DB_CACHE_SIZE = 500M
- LARGE_POOL_SIZE=200M

Also assume that the result of the queries is as follows:

```
SELECT SUM(value) FROM V$SGA = 1200M
SELECT CURRENT_VALUE FROM V$SGA_DYNAMIC_FREE_MEMORY = 208M
```

You can take advantage of automatic shared memory management by replacing the manually sized parameters (SHARED_POOL_SIZE, DB_CACHE_SIZE, LARGE_POOL_SIZE) with the following setting:

```
SGA_TARGET = 992M
```

where 992M = 1200M minus 208M.

By default, when you set a value for SGA_TARGET, the value of the parameters corresponding to all the automatically sized SGA components is set to zero. However, you can still exercise some control over the size of the automatically sized components by specifying minimum values for the parameters corresponding to these components. Doing so can be useful if you know that an application cannot function properly without a minimum amount of memory in specific components. You specify the minimum amount of SGA space for a component by setting a value for its corresponding initialization parameter. Here is an example configuration:

- SGA_TARGET = 256M
- SHARED_POOL_SIZE = 32M
- DB_CACHE_SIZE = 100M

In this example, the shared pool and the default buffer pool will not be sized smaller than the specified values (32 M and 100M, respectively). The remaining 124M (256 minus 132) is available for use by all the manually and automatically sized components.

The actual distribution of values among the SGA components might look like this:

- Actual shared pool size = 64M
- Actual buffer cache size = 128M
- Actual Java pool size = 60M
- Actual large pool size = 4M

The parameter values determine the minimum amount of SGA space allocated. The fixed view V$SGA_DYNAMIC_COMPONENTS displays the current actual size of each SGA component. You can also see the current actual values of the SGA components in the Enterprise Manager memory configuration page.

Manually limiting the minimum size of one or more automatically sized components reduces the total amount of memory available for dynamic adjustment. This reduction in turn limits the ability of the system to adapt to workload changes. Therefore, this practice is not recommended except in exceptional cases. The default automatic management behavior maximizes both system performance and the use of available resources.

When the automatic shared memory management feature is enabled, the internal tuning algorithm tries to determine an optimal size for the shared pool based on the workload. It usually converges on this value by increasing in small increments over time. However, the internal tuning algorithm typically does not attempt to shrink the shared pool, because the presence of open cursors, pinned PL/SQL packages, and other SQL execution state in the shared pool make it impossible to find granules that can be freed. Therefore, the tuning algorithmn only tries to increase

the shared pool in conservative increments, starting from a conservative size and stabilizing the shared pool at a size that produces the optimal performance benefit.

Dynamic Modification of SGA Parameters You can modify the value of SGA_TARGET and the parameters controlling individual components dynamically using the ALTER SYSTEM statement, as described in the sections that follow.

Dynamic Modification of SGA_TARGET The SGA_TARGET parameter can be increased up to the value specified for the SGA_MAX_SIZE parameter, and it can also be reduced. If you reduce the value of SGA_TARGET, the system identifies one or more automatically tuned components to release memory. You can reduce SGA_TARGET until one or more automatically tuned components reach their minimum size. Oracle Database determines the minimum allowable value for SGA_TARGET taking into account several factors, including values set for the automatically sized components, manually sized components that use SGA_TARGET space, and number of CPUs.

The change in the amount of physical memory consumed when SGA_TARGET is modified depends on the operating system. On some UNIX platforms that do not support dynamic shared memory, the physical memory in use by the SGA is equal to the value of the SGA_MAX_SIZE parameter. On such platforms, there is no real benefit in setting SGA_TARGET to a value smaller than SGA_MAX_SIZE. Therefore, setting SGA_MAX_SIZE on those platforms is not recommended.

On other platforms, such as Solaris and Windows, the physical memory consumed by the SGA is equal to the value of SGA_TARGET.

When SGA_TARGET is resized, the only components affected are the automatically tuned components for which you have not set a minimum value in their corresponding initialization parameter. Any manually configured components remain unaffected. For example, suppose you have an environment with the following configuration:

- SGA_MAX_SIZE = 1024M
- SGA_TARGET = 512M
- DB_8K_CACHE_SIZE = 128M

In this example, the value of SGA_TARGET can be resized up to 1024M and can also be reduced until one or more of the buffer cache, shared pool, large pool, or java pool reaches its minimum size. The exact value depends on environmental factors such as the number of CPUs on the system. However, the value of DB_8K_CACHE_SIZE remains fixed at all times at 128M

When SGA_TARGET is reduced, if the values for any automatically tuned component sizes has been specified to limit their minimum sizes, then those components will not be reduced smaller than that minimum. Consider the following combination of parameters:

- SGA_MAX_SIZE = 1024M
- SGA_TARGET = 512M
- DB_CACHE_SIZE = 96M
- DB_8K_CACHE_SIZE = 128M

As in the last example, if SGA_TARGET is reduced, the DB_8K_CACHE_SIZE parameter is permanently fixed at 128M. In addition, the primary buffer cache (determined by the DB_CACHE_SIZE parameter) will not be reduced smaller than 96M. Thus the amount that SGA_TARGET can be reduced is restricted.

Modifying Parameters for Automatically Managed Components When SGA_TARGET is not set, the automatic shared memory management feature is not enabled. Therefore the rules governing resize for all component parameters are the same as in earlier releases. However, when automatic shared memory management is enabled, the manually specified sizes of automatically sized components serve as a lower bound for the size of the components. You can modify this limit dynamically by changing the values of the corresponding parameters. For example, consider the following configuration:

- SGA_TARGET = 512M
- LARGE_POOL_SIZE = 256M
- Current actual shared pool size = 284M

In this example, if you increase the value of LARGE_POOL_SIZE to a value greater than the actual current size of the component, the system expands the component to accommodate the increased minimum size. For example, if you increase the value of LARGE_POOL_SIZE to 300M, then the system increases the large pool incrementally until it reaches 300M. This resizing occurs at the expense of one or more automatically tuned components.

If you decrease the value of LARGE_POOL_SIZE to 200, there is no immediate change in the size of that component. The new setting only limits the reduction of the large pool size to 200 M in the future.

Modifying Parameters for Manually Sized Components Parameters for manually sized components can be dynamically altered as well. However, rather than setting a minimum size, the value of the parameter specifies the precise size of the corresponding component. When you increase the size of a manually sized

component, extra memory is taken away from one or more automatically sized components. When you decrease the size of a manually sized component, the memory that is released is given to the automatically sized components.

For example, consider this configuration:

- SGA_TARGET = 512M
- DB_8K_CACHE_SIZE = 128M

In this example, increasing DB_8K_CACHE_SIZE by 16 M to 144M means that the 16M will be taken away from the automatically sized components. Likewise, reducing DB_8K_CACHE_SIZE by 16M to 112M means that the 16M will be given to the automatically sized components.

Using Manual Shared Memory Management

If you decide not to use automatic shared memory management by not setting the SGA_TARGET parameter, you must manually configure each component of the SGA. This section provides guidelines on setting the parameters that control the size of each SGA components.

Setting the Buffer Cache Initialization Parameters The buffer cache initialization parameters determine the size of the buffer cache component of the SGA. You use them to specify the sizes of caches for the various block sizes used by the database. These initialization parameters are all dynamic.

If you intend to use multiple block sizes in your database, you must have the DB_CACHE_SIZE and at least one DB_*n*K_CACHE_SIZE parameter set. Oracle Database assigns an appropriate default value to the DB_CACHE_SIZE parameter, but the DB_*n*K_CACHE_SIZE parameters default to 0, and no additional block size caches are configured.

The size of a buffer cache affects performance. Larger cache sizes generally reduce the number of disk reads and writes. However, a large cache may take up too much memory and induce memory paging or swapping.

DB_CACHE_SIZE Initialization Parameter The DB_CACHE_SIZE initialization parameter has replaced the DB_BLOCK_BUFFERS initialization parameter, which was used in earlier releases. The DB_CACHE_SIZE parameter specifies the size in bytes of the cache of standard block size buffers. Thus, to specify a value for DB_CACHE_SIZE, you would determine the number of buffers that you need and multiple that value times the block size specified in DB_BLOCK_SIZE.

For backward compatibility, the DB_BLOCK_BUFFERS parameter still functions, but it remains a static parameter and cannot be combined with any of the dynamic sizing parameters.

The DB_nK_CACHE_SIZE Initialization Parameters The sizes and numbers of nonstandard block size buffers are specified by the following initialization parameters:

- DB_2K_CACHE_SIZE
- DB_4K_CACHE_SIZE
- DB_8K_CACHE_SIZE
- DB_16K_CACHE_SIZE
- DB_32K_CACHE_SIZE

Each parameter specifies the size of the buffer cache for the corresponding block size. For example:

```
DB_BLOCK_SIZE=4096

DB_CACHE_SIZE=12M
DB_2K_CACHE_SIZE=8M
DB_8K_CACHE_SIZE=4M
```

In this example, the parameters specify that the standard block size of the database is 4K. The size of the cache of standard block size buffers is 12M. Additionally, 2K and 8K caches will be configured with sizes of 8M and 4M respectively.

Note: You cannot use a DB_nK_CACHE_SIZE parameter to size the cache for the standard block size. For example, if the value of DB_BLOCK_SIZE is 2K, it is invalid to set DB_2K_CACHE_SIZE. The size of the cache for the standard block size is always determined from the value of DB_CACHE_SIZE.

Specifying the Shared Pool Size The SHARED_POOL_SIZE initialization parameter is a dynamic parameter that lets you specify or adjust the size of the shared pool component of the SGA. Oracle Database selects an appropriate default value. Configuring the shared pool is discussed in *Oracle Database Performance Tuning Guide*.

Specifying the Large Pool Size The LARGE_POOL_SIZE initialization parameter is a dynamic parameter that lets you specify or adjust the size of the large pool component of the SGA. The large pool is an optional component of the SGA. You must specifically set the LARGE_POOL_SIZE parameter if you want to create a large pool. Configuring the large pool is discussed in *Oracle Database Performance Tuning Guide*.

Specifying the Java Pool Size The JAVA_POOL_SIZE initialization parameter is a dynamic parameter that lets you specify or adjust the size of the java pool component of the SGA. Oracle Database selects an appropriate default value. Configuration of the java pool is discussed in *Oracle Database Java Developer's Guide*.

Specifying the Streams Pool Size The STREAMS_POOL_SIZE initialization parameter is a dynamic parameter that lets you specify or adjust the size of the Streams pool component of the SGA. If STREAMS_POOL_SIZE is set to 0, then the Oracle Streams product will use the shared pool to satisfy its SGA memory requirements. Configuration of the Streams pool is discussed in *Oracle Streams Concepts and Administration*.

Viewing Information About the SGA

The following views provide information about the SGA components and their dynamic resizing:

View	Description
V$SGA	Displays summary information about the system global area (SGA).
V$SGAINFO	Displays size information about the SGA, including the sizes of different SGA components, the granule size, and free memory.
V$SGASTAT	Displays detailed information about the SGA.
V$SGA_DYNAMIC_COMPONENTS	Displays information about the dynamic SGA components. This view summarizes information based on all completed SGA resize operations since instance startup.
V$SGA_DYNAMIC_FREE_MEMORY	Displays information about the amount of SGA memory available for future dynamic SGA resize operations.
V$SGA_RESIZE_OPS	Displays information about the last 400 completed SGA resize operations.

View	Description
V$SGA_CURRENT_RESIZE_OPS	Displays information about SGA resize operations that are currently in progress. A resize operation is an enlargement or reduction of a dynamic SGA component.

Specifying the Maximum Number of Processes

The PROCESSES initialization parameter determines the maximum number of operating system processes that can be connected to Oracle Database concurrently. The value of this parameter must be a minimum of one for each background process plus one for each user process. The number of background processes will vary according the database features that you are using. For example, if you are using Advanced Queuing or the file mapping feature, you will have additional background processes. If you are using Automatic Storage Management, then add three additional processes.

If you plan on running 50 user processes, a good estimate would be to set the PROCESSES initialization parameter to 60.

Specifying the Method of Undo Space Management

Every Oracle Database must have a method of maintaining information that is used to roll back, or undo, changes to the database. Such information consists of records of the actions of transactions, primarily before they are committed. Collectively these records are called **undo data**.

The use of manual undo management mode, which stores undo data in rollback segments, has been deprecated. This section provides instructions for setting up an environment for automatic undo management using an undo tablespace. An undo tablespace is easier to administer than rollback segments, and automatic undo management lets you explicitly set a retention guarantee time.

See Also: Chapter 10, "Managing the Undo Tablespace"

UNDO_MANAGEMENT Initialization Parameter

The UNDO_MANAGEMENT initialization parameter determines whether an instance will start up in automatic undo management mode, which stores undo in an undo tablespace, or manual undo management mode, which stores undo in rollback segments. By default, this parameter is set to MANUAL. Set this parameter to AUTO to enable automatic undo management mode.

UNDO_TABLESPACE Initialization Parameter

When an instance starts up in automatic undo management mode, it attempts to select an undo tablespace for storage of undo data. If the database was created in undo management mode, then the default undo tablespace--either the system-created SYS_UNDOTS tablespace or the user-specified undo tablespace--is the undo tablespace used at instance startup. You can override this default for the instance by specifying a value for the UNDO_TABLESPACE initialization parameter. This parameter is especially useful for assigning a particular undo tablespace to an instance in an Oracle Real Application Clusters environment.

If no undo tablespace has been specified during database creation or by the UNDO_TABLESPACE initialization parameter, then the instance will start. However, it uses the SYSTEM rollback segment for storage of undo data. This use of the SYSTEM rollback segment is not recommended in normal circumstances, and an alert message is written to the alert file to warn that the system is running without an undo tablespace. ORA-01552 errors are issued for any attempts to write non-SYSTEM related undo data to the SYSTEM rollback segment.

The COMPATIBLE Initialization Parameter and Irreversible Compatibility

The COMPATIBLE initialization parameter enables or disables the use of features in the database that affect file format on disk. For example, if you create an Oracle Database 10*g* database, but specify `COMPATIBLE = 9.2.0.2` in the initialization parameter file, then features that requires 10.0 compatibility will generate an error if you try to use them. Such a database is said to be at the 9.2.0.2 compatibility level.

You can advance the compatibility level of your database. If you do advance the compatibility of your database with the COMPATIBLE initialization parameter, then there is no way to start the database using a lower compatibility level setting, except by doing a point-in-time recovery to a time before the compatibility was advanced.

The default value for the COMPATIBLE parameter is the release number of the most recent major release.

Caution: For Oracle Database 10*g* Release 1 (10.1), the default value of the COMPATIBLE parameter is 10.0.0. If you create an Oracle Database using this default value, you can immediately use all the new features in this release, and you can never downgrade the database.

See Also:

- *Oracle Database Upgrade Guide* for a detailed discussion of database compatibility and the `COMPATIBLE` initialization parameter
- *Oracle Database Backup and Recovery Advanced User's Guide* for information about point-in-time recovery of your database

Setting the License Parameter

Note: Oracle no longer offers licensing by the number of concurrent sessions. Therefore the `LICENSE_MAX_SESSIONS` and `LICENSE_SESSIONS_WARNING` initialization parameters are no longer needed and have been deprecated.

If you use named user licensing, Oracle Database can help you enforce this form of licensing. You can set a limit on the number of users created in the database. Once this limit is reached, you cannot create more users.

Note: This mechanism assumes that each person accessing the database has a unique user name and that no people share a user name. Therefore, so that named user licensing can help you ensure compliance with your Oracle license agreement, do not allow multiple users to log in using the same user name.

To limit the number of users created in a database, set the `LICENSE_MAX_USERS` initialization parameter in the database initialization parameter file, as shown in the following example:

```
LICENSE_MAX_USERS = 200
```

Troubleshooting Database Creation

If database creation fails, you can look at the alert log to determine the reason for the failure and to determine corrective action. The alert log is discussed in "Monitoring the Operation of Your Database" on page 4-25.

You should shut down the instance and delete any files created by the CREATE DATABASE statement before you attempt to create it again. After correcting the error that caused the failure of the database creation, try re-creating the database.

Dropping a Database

Dropping a database involves removing its datafiles, redo log files, control files, and initialization parameter files. The DROP DATABASE statement deletes all control files and all other database files listed in the control file. To use the DROP DATABASE statement successfully, all of the following conditions must apply:

- The database must be mounted and closed.
- The database must be mounted exclusively--not in shared mode.
- The database must be mounted as RESTRICTED.

An example of this statement is:

```
DROP DATABASE;
```

The DROP DATABASE statement has no effect on archived log files, nor does it have any effect on copies or backups of the database. It is best to use RMAN to delete such files. If the database is on raw disks, the actual raw disk special files are not deleted.

If you used the Database Configuration Assistant to create your database, you can use that tool to delete (drop) your database and remove the files.

Managing Initialization Parameters Using a Server Parameter File

Initialization parameters for the Oracle Database have traditionally been stored in a text initialization parameter file. For better manageability, you can choose to maintain initialization parameters in a binary server parameter file that is persistent across database startup and shutdown. This section introduces the server parameter file, and explains how to manage initialization parameters using either method of storing the parameters. The following topics are contained in this section.

- What Is a Server Parameter File?
- Migrating to a Server Parameter File
- Creating a Server Parameter File
- The SPFILE Initialization Parameter

What Is a Server Parameter File?

A **server parameter file** (SPFILE) can be thought of as a repository for initialization parameters that is maintained on the machine running the Oracle Database server. It is, by design, a server-side initialization parameter file. Initialization parameters stored in a server parameter file are persistent, in that any changes made to the parameters while an instance is running can persist across instance shutdown and startup. This arrangement eliminates the need to manually update initialization parameters to make changes effected by ALTER SYSTEM statements persistent. It also provides a basis for self-tuning by the Oracle Database server.

A server parameter file is initially built from a traditional text initialization parameter file using the CREATE SPFILE statement. It is a binary file that cannot be edited using a text editor. Oracle Database provides other interfaces for viewing and modifying parameter settings.

Caution: Although you can open the binary server parameter file with a text editor and view its text, *do not* manually edit it. Doing so will corrupt the file. You will not be able to start your instance, and if the instance is running, it could fail.

At system startup, the default behavior of the STARTUP command is to read a server parameter file to obtain initialization parameter settings. The STARTUP command with no PFILE clause reads the server parameter file from an operating system specific location. If you use a traditional text initialization parameter file, you must specify the PFILE clause when issuing the STARTUP command. Instructions for starting an instance using a server parameter file are contained in "Starting Up a Database" on page 3-1.

Migrating to a Server Parameter File

If you are currently using a traditional initialization parameter file, use the following steps to migrate to a server parameter file.

1. If the initialization parameter file is located on a client machine, transfer the file (for example, FTP) from the client machine to the server machine.

   Note: If you are migrating to a server parameter file in an Oracle Real Application Clusters environment, you must combine all of your instance-specific initialization parameter files into a single initialization parameter file. Instructions for doing this, and other actions unique to using a server parameter file for instances that are part of an Oracle Real Application Clusters installation, are discussed in:

   - *Oracle Real Application Clusters Installation and Configuration Guide*
   - *Oracle Real Application Clusters Administrator's Guide*

2. Create a server parameter file using the CREATE SPFILE statement. This statement reads the initialization parameter file to create a server parameter file. The database does not have to be started to issue a CREATE SPFILE statement.
3. Start up the instance using the newly created server parameter file.

Creating a Server Parameter File

The server parameter file is initially created from a traditional text initialization parameter file. It must be created prior to its use in the STARTUP command. The CREATE SPFILE statement is used to create a server parameter file. You must have the SYSDBA or the SYSOPER system privilege to execute this statement.

The following example creates a server parameter file from initialization parameter file /u01/oracle/dbs/init.ora. In this example no SPFILE name is specified, so the file is created in a platform-specific default location and is named spfile$ORACLE_SID.ora.

```
CREATE SPFILE FROM PFILE='/u01/oracle/dbs/init.ora';
```

Another example, which follows, illustrates creating a server parameter file and supplying a name.

```
CREATE SPFILE='/u01/oracle/dbs/test_spfile.ora'
       FROM PFILE='/u01/oracle/dbs/test_init.ora';
```

When you create a server parameter file from an initialization parameter file, comments specified on the same lines as a parameter setting in the initialization parameter file are maintained in the server parameter file. All other comments are ignored.

The server parameter file is always created on the machine running the database server. If a server parameter file of the same name already exists on the server, it is overwritten with the new information.

Oracle recommends that you allow the database server to default the name and location of the server parameter file. This will ease administration of your database. For example, the STARTUP command assumes this default location to read the parameter file. The table that follows shows the default name and location of the server parameter file. The table assumes that the SPFILE is a file. If it is a raw device, the default name could be a logical volume name or partition device name, and the default location could differ.

Platform	Default Name	Default Location
UNIX	spfile*dbname*.ora	*$ORACLE_HOME*/dbs or the same location as the datafiles
Windows	spfile*SID*.ora	*ORACLE_BASE**ORACLE_HOME*\database

If you create a server parameter file in a location other than the default location, then you must create a parameter file with the default parameter file name in the default location that points to the user-generated SPFILE. The parameter file would contain one line:

```
SPFILE = 'location'
```

The CREATE SPFILE statement can be executed before or after instance startup. However, if the instance has been started using a server parameter file, an error is raised if you attempt to re-create the same server parameter file that is currently being used by the instance.

Note: When you use the Database Configuration Assistant to create a database, it can automatically create a server parameter file for you.

The SPFILE Initialization Parameter

The SPFILE initialization parameter contains the name of the current server parameter file. When the default server parameter file is used by the server (that is, you issue a STARTUP command and do not specify a PFILE), the value of SPFILE is internally set by the server. The SQL*Plus command SHOW PARAMETERS SPFILE (or any other method of querying the value of a parameter) displays the name of the server parameter file that is currently in use.

The SPFILE parameter can also be set in a traditional parameter file to indicate the server parameter file to use. You use the SPFILE parameter to specify a server parameter file located in a nondefault location. *Do not* use an IFILE initialization parameter within a traditional initialization parameter file to point to a server parameter file; instead, use the SPFILE parameter. See "Starting Up a Database" on page 3-1 for details about:

- Starting up a database that uses a server parameter file
- Using the SPFILE parameter to specify the name of a server parameter file to use at instance startup

Using ALTER SYSTEM to Change Initialization Parameter Values

The ALTER SYSTEM statement lets you set, change, or restore to default the values of initialization parameter. If you are using a traditional initialization parameter file, the ALTER SYSTEM statement changes the value of a parameter only for the current instance, because there is no mechanism for automatically updating initialization parameters on disk. You must update them manually to be passed to a future instance. Using a server parameter file overcomes this limitation.

Setting or Changing Initialization Parameter Values

Use the SET clause of the ALTER SYSTEM statement to set or change initialization parameter values. The optional SCOPE clause specifies the scope of a change as described in the following table:

SCOPE Clause	Description
SCOPE = SPFILE	The change is applied in the server parameter file only. The effect is as follows: ■ For dynamic parameters, the change is effective at the next startup and is persistent. ■ For static parameters, the behavior is the same as for dynamic parameters. This is the only SCOPE specification allowed for static parameters.
SCOPE = MEMORY	The change is applied in memory only. The effect is as follows: ■ For dynamic parameters, the effect is immediate, but it is not persistent because the server parameter file is not updated. ■ For static parameters, this specification is not allowed.
SCOPE = BOTH	The change is applied in both the server parameter file and memory. The effect is as follows: ■ For dynamic parameters, the effect is immediate and persistent. ■ For static parameters, this specification is not allowed.

It is an error to specify SCOPE=SPFILE or SCOPE=BOTH if the server is not using a server parameter file. The default is SCOPE=BOTH if a server parameter file was used to start up the instance, and MEMORY if a traditional initialization parameter file was used to start up the instance.

For dynamic parameters, you can also specify the DEFERRED keyword. When specified, the change is effective only for future sessions.

An optional COMMENT clause lets you associate a text string with the parameter update. When you specify SCOPE as SPFILE or BOTH, the comment is written to the server parameter file.

The following statement changes the maximum number of job queue processes allowed for the instance. It includes a comment, and explicitly states that the change is to be made only in memory (that is, it is not persistent across instance shutdown and startup).

```
ALTER SYSTEM SET JOB_QUEUE_PROCESSES=50
                 COMMENT='temporary change on Nov 29'
                 SCOPE=MEMORY;
```

The next example sets a complex initialization parameter that takes a list of attributes. Specifically, the parameter value being set is the LOG_ARCHIVE_DEST_n initialization parameter. This statement could change an existing setting for this parameter of create a new archive destination.

```
ALTER SYSTEM
     SET LOG_ARCHIVE_DEST_4='LOCATION=/u02/oracle/rbdb1/',MANDATORY,'REOPEN=2'
         COMMENT='Add new destimation on Nov 29'
         SCOPE=SPFILE;
```

When a value consists of a list of parameters, you cannot edit individual attributes by the position or ordinal number. You must specify the complete list of values each time the parameter is updated, and the new list completely replaces the old list.

Deleting Initialization Parameter Values

If an initialization parameter takes a value that is a string, you can restore the value to its default value by using empty single quote marks, as follows:

```
ALTER SYSTEM SET parameter = '';
```

For numeric and boolean value parameters, you must explicitly set the parameter back to its original default value.

Exporting the Server Parameter File

You can use the CREATE PFILE statement to export a server parameter file to a traditional text initialization parameter file. Doing so might be necessary for several reasons:

- To create backups of the server parameter file
- For diagnostic purposes, listing all of the parameter values currently used by an instance. This is analogous to the SQL*Plus SHOW PARAMETERS command or selecting from the V$PARAMETER or V$PARAMETER2 views.
- To modify the server parameter file by first exporting it, editing the resulting text file, and then re-creating it using the CREATE SPFILE statement

The exported file can also be used to start up an instance using the PFILE clause.

You must have the SYSDBA or the SYSOPER system privilege to execute the CREATE PFILE statement. The exported file is created on the database server machine. It contains any comments associated with the parameter in the same line as the parameter setting.

The following example creates a text initialization parameter file from the server parameter file:

```
CREATE PFILE FROM SPFILE;
```

Because no names were specified for the files, the database creates an initialization parameter file with a platform-specific name, and it is created from the platform-specific default server parameter file.

The following example creates a text initialization parameter file from a server parameter file, but in this example the names of the files are specified:

```
CREATE PFILE='/u01/oracle/dbs/test_init.ora'
       FROM SPFILE='/u01/oracle/dbs/test_spfile.ora';
```

Backing Up the Server Parameter File

You can create a backup of your server parameter file by exporting it, as described in "Exporting the Server Parameter File" on page 2-51. If the backup and recovery strategy for your database is implemented using Recovery Manager (RMAN), then you can use RMAN to create a backup. The server parameter file is backed up automatically by RMAN when you back up your database, but RMAN also lets you specifically create a backup of the currently active server parameter file.

See Also: *Oracle Database Backup and Recovery Basics*

Errors and Recovery for the Server Parameter File

If an error occurs while reading the server parameter file (during startup or an export operation), or while writing the server parameter file during its creation, the operation terminates with an error reported to the user.

If an error occurs while reading or writing the server parameter file during a parameter update, the error is reported in the alert file and all subsequent parameter updates to the server parameter file are ignored. At this point, you can take one of the following actions:

- Shut down the instance, recover the server parameter file, and then restart the instance.
- Continue to run the database if you do not care that subsequent parameter updates will not be persistent.

Viewing Parameter Settings

You can view parameter settings in several ways, as shown in the following table.

Method	Description
SHOW PARAMETERS	This SQL*Plus command displays the values of parameters currently in use.
CREATE PFILE	This SQL statement creates a text initialization parameter file from the binary server parameter file.
V$PARAMETER	This view displays the values of parameters currently in effect.
V$PARAMETER2	This view displays the values of parameters currently in effect. It is easier to distinguish list parameter values in this view because each list parameter value appears as a row.
V$SPPARAMETER	This view displays the current contents of the server parameter file. The view returns FALSE values in the ISSPECIFIED column if a server parameter file is not being used by the instance.

See Also: *Oracle Database Reference* for a complete description of views

Defining Application Services for Oracle Database 10*g*

This section describes Oracle Database 10*g* services and includes the following topics:

- Deploying Services
- Configuring Services
- Using Services

Services are logical abstractions for managing workloads in Oracle Database 10*g*. Services divide workloads into mutually disjoint groupings. Each service represents a workload with common attributes, service-level thresholds, and priorities. The grouping is based on attributes of work that might include the application function to be used, the priority of execution for the application function, the job class to be managed, or the data range used in the application function or job class. For example, the Oracle E-Business suite defines a service for each responsibility, such as general ledger, accounts receivable, order entry, and so on.

In Oracle Database 10*g*, services are built into the Oracle Database providing single system image for workloads, prioritization for workloads, performance measures for real transactions, and alerts and actions when performance goals are violated. Services enable you to configure a workload, administer it, enable and disable it, and measure the workload as a single entity. You can do this using standard tools such as the Database Configuration Assistant (DBCA), Net Configuration Assistant (NetCA), and Enterprise Manager (EM). Enterprise Manager supports viewing and operating services as a whole, with drill down to the instance-level when needed.

In Real Application Clusters (RAC), a service can span one or more instances and facilitate real workload balancing based on real transaction performance. This provides end-to-end unattended recovery, rolling changes by workload, and full location transparency. RAC also enables you to manage a number of service features with Enterprise Manager, the DBCA, and the Server Control utility (SRVCTL).

Services also offer an extra dimension in performance tuning. Tuning by "service and SQL" can replace tuning by "session and SQL" in the majority of systems where all sessions are anonymous and shared. With services, workloads are visible and measurable. Resource consumption and waits are attributable by application. Additionally, resources assigned to services can be augmented when loads increase or decrease. This dynamic resource allocation enables a cost-effective solution for meeting demands as they occur. For example, services are measured automatically and the performance is compared to service-level thresholds. Performance violations are reported to Enterprise Manager, enabling the execution of automatic or scheduled solutions.

> **See Also:** *Oracle Real Application Clusters Deployment and Performance Guide* for more information about services in RAC

Deploying Services

Installations configure Oracle Database 10*g* services in the database giving each service a unique global name, associated performance goals, and associated importance. The services are tightly integrated with the Oracle Database and are maintained in the data dictionary. You can find service information in the following service-specific views:

- `DBA_SERVICES`
- `ALL_SERVICES` or `V$SERVICES`
- `V$ACTIVE_SERVICES`
- `V$SERVICE_STATS`

- V$SERVICE_EVENTS
- V$SERVICE_WAIT_CLASSES
- V$SERV_MOD_ACT_STATS
- V$SERVICE_METRICS
- V$SERVICE_METRICS_HISTORY

The following additional views also contain some information about services:

- V$SESSION
- V$ACTIVE_SESSION_HISTORY
- DBA_RSRC_GROUP_MAPPINGS
- DBA_SCHEDULER_JOB_CLASSES
- DBA_THRESHOLDS

> **See Also:** *Oracle Database Reference* for detailed information about these views

Several Oracle Database features support services. The Automatic Workload Repository (AWR) manages the performance of services. AWR records service performance, including execution times, wait classes, and resources consumed by service. AWR alerts warn when service response time thresholds are exceeded. The dynamic views report current service performance metrics with one hour of history. Each service has quality-of-service thresholds for response time and CPU consumption.

In addition, the Database Resource Manager maps services to consumer groups. This enables you to automatically manage the priority of one service relative to others. You can use consumer groups to define relative priority in terms of either ratios or resource consumption. This is described in more detail, for example, in *Oracle Real Application Clusters Deployment and Performance Guide*.

Configuring Services

Services describe applications, application functions, and data ranges as either functional services or data-dependent services. Functional services are the most common mapping of workloads. Sessions using a particular function are grouped together. For Oracle*Applications, ERP, CRM, and *i*Support functions create a functional division of the work. For SAP, dialog and update functions create a functional division of the work.

In contrast, data-dependent routing routes sessions to services based on data keys. The mapping of work requests to services occurs in the object relational mapping layer for application servers and TP monitors. For example, in RAC, these ranges can be completely dynamic and based on demand because the database is shared.

You can also define preconnect application services in RAC databases. Preconnect services span instances to support a service in the event of a failure. The preconnect service supports TAF preconnect mode and is managed transparently when using RAC.

In addition to application services, Oracle Database also supports two internal services: SYS$BACKGROUND is used by the background processes only and SYS$USERS is the default service for user sessions that are not associated with services.

Use the DBMS_SERVICE package or set the SERVICE_NAMES parameter to create application services on a single-instance Oracle Database. You can later define the response time goal or importance of each service through EM, either individually or by using the Enterprise Manager feature "Copy Thresholds From a Baseline" on the Manage Metrics / Edit Threshold pages. You can also do this using PL / SQL.

Using Services

Using services requires no changes to your application code. Client-side work connects to a service. Server-side work specifies the service when creating the job class for the Job Scheduler and the database links for distributed databases. Work requests executing under a service inherit the performance thresholds for the service and are measured as part of the service.

Client-side Use

Middle-tier applications and client-server applications use a service by specifying the service as part of the connection in TNS connect data. This connect data may be in the TNSnames file for thick Net drivers, in the URL specification for thin drivers, or may be maintained in the Oracle Internet Directory. For example, data sources for the Oracle Application Server 10*g* are set to route to a service. Using Net Easy*Connection in Oracle Database 10*g*, this connection needs only the service and network address, for example, `hr/hr@//myVIP/myservice`. For Oracle E-Business Suite, the service is also maintained in the application database identifier and in the cookie for the ICX parameters.

Server-side Use

Server-side work, such as the Job Scheduler, parallel execution, and Oracle Streams Advanced Queuing, set the service name as part of the workload definition.

For the Job Scheduler, the service that the job class uses is defined when the job class is created. During execution, jobs are assigned to job classes, and job classes run within services. Using services with job classes ensures that the work executed by the job scheduler is identified for workload management and performance tuning.

For parallel query and parallel DML, the query coordinator connects to a service just like any other client. The parallel query processes inherit the service for the duration of the execution. At the end of query execution, the parallel execution processes revert to the default service.

Considerations After Creating a Database

After you create a database, the instance is left running, and the database is open and available for normal database use. You may want to perform other actions, some of which are discussed in this section.

Some Security Considerations

Note Regarding Security Enhancements: In this release of Oracle Database and in subsequent releases, several enhancements are being made to ensure the security of default database user accounts. You can find a security checklist for this release in *Oracle Database Security Guide*. Oracle recommends that you read this checklist and configure your database accordingly.

Once the database is created, you can configure it to take advantage of Oracle Identity Management. For information on how to do this, please refer to *Oracle Advanced Security Administrator's Guide*.

A newly created database has at least three user accounts that are important for administering your database: SYS, SYSTEM, and SYSMAN.

Caution: To prevent unauthorized access and protect the integrity of your database, it is important that new passwords for user accounts SYS and SYSTEM be specified when the database is created. This is accomplished by specifying the following CREATE DATABASE clauses when manually creating you database, or by using DBCA to create the database:

- `USER SYS IDENTIFIED BY`
- `USER SYSTEM IDENTIFIED BY`

Additional administrative accounts are provided by Oracle Database that should be used only by authorized users. To protect these accounts from being used by unauthorized users familiar with their Oracle-supplied passwords, these accounts are initially locked with their passwords expired. As the database administrator, you are responsible for the unlocking and resetting of these accounts.

Table 2–5 lists the administrative accounts that are provided by Oracle Database. Not all accounts may be present on your system, depending upon the options that you selected for your database.

Table 2–5 Administrative User Accounts Provided by Oracle Database

Username	Password	Description	See Also
CTXSYS	CTXSYS	The Oracle Text account	*Oracle Text Reference*
DBSNMP	DBSNMP	The account used by the Management Agent component of Oracle Enterprise Manager to monitor and manage the database	*Oracle Enterprise Manager Grid Control Installation and Basic Configuration*
LBACSYS	LBACSYS	The Oracle Label Security administrator account	*Oracle Label Security Administrator's Guide*
MDDATA	MDDATA	The schema used by Oracle Spatial for storing Geocoder and router data	*Oracle Spatial User's Guide and Reference*
MDSYS	MDSYS	The Oracle Spatial and Oracle *inter*Media Locator administrator account	*Oracle Spatial User's Guide and Reference*
DMSYS	DMSYS	The Oracle Data Mining account.	*Oracle Data Mining Administrator's Guide* *Oracle Data Mining Concepts*

Table 2–5 (Cont.) Administrative User Accounts Provided by Oracle Database

Username	Password	Description	See Also
OLAPSYS	MANAGER	The account used to create OLAP metadata structures. It owns the OLAP Catalog (CWMLite).	*Oracle OLAP Application Developer's Guide*
ORDPLUGINS	ORDPLUGINS	The Oracle *inter*Media user. Plug-ins supplied by Oracle and third party format plug-ins are installed in this schema.	*Oracle interMedia User's Guide*
ORDSYS	ORDSYS	The Oracle *inter*Media administrator account	*Oracle interMedia User's Guide*
OUTLN	OUTLN	The account that supports plan stability. Plan stability enables you to maintain the same execution plans for the same SQL statements. OUTLN acts as a role to centrally manage metadata associated with stored outlines.	*Oracle Database Performance Tuning Guide*
SI_INFORMTN_ SCHEMA	SI_INFORMTN_ SCHEMA	The account that stores the information views for the SQL / MM Still Image Standard	*Oracle interMedia User's Guide*
SYS	CHANGE_ON_ INSTALL	The account used to perform database administration tasks	*Oracle Database Administrator's Guide*
SYSMAN	CHANGE_ON_ INSTALL	The account used to perform Oracle Enterprise Manager database administration tasks. Note that SYS and SYSTEM can also perform these tasks.	*Oracle Enterprise Manager Grid Control Installation and Basic Configuration*
SYSTEM	MANAGER	Another account used to perform database administration tasks.	*Oracle Database Administrator's Guide*

See Also:

- "Database Administrator Usernames" on page 1-10 for more information about the users SYS and SYSTEM
- *Oracle Database Security Guide* to learn how to add new users and change passwords
- *Oracle Database SQL Reference* for the syntax of the ALTER USER statement used for unlocking user accounts

Installing the Oracle Database Sample Schemas

The Oracle Database distribution media can includes various SQL files that let you experiment with the system, learn SQL, or create additional tables, views, or synonyms.

Oracle Database includes sample schemas that help you to become familiar with Oracle Database functionality. All Oracle Database documentation and training materials are being converted to the Sample Schemas environment as those materials are updated.

The Sample Schemas can be installed automatically by the Database Configuration Assistant, or you can install them manually. The schemas and installation instructions are described in detail in *Oracle Database Sample Schemas*.

Viewing Information About the Database

In addition to the views listed previously in "Viewing Parameter Settings", you can view information about your database content and structure using the following views:

View	Description
DATABASE_PROPERTIES	Displays permanent database properties
GLOBAL_NAME	Displays the global database name
V$DATABASE	Contains database information from the control file

3

Starting Up and Shutting Down

This chapter describes the procedures for starting up and shutting down an Oracle Database instance and contains the following topics:

- Starting Up a Database
- Altering Database Availability
- Shutting Down a Database
- Quiescing a Database
- Suspending and Resuming a Database

> **See Also:** *Oracle Real Application Clusters Administrator's Guide* for additional information specific to an Oracle Real Application Clusters environment

Starting Up a Database

When you start up a database, you create an instance of that database and you determine the state of the database. Normally, you start up an instance by mounting and opening the database. Doing so makes the database available for any valid user to connect to and perform typical data access operations. Other options exist, and these are also discussed in this section.

This section contains the following topics relating to starting up an instance of a database:

- Options for Starting Up a Database
- Preparing to Start an Instance
- Using SQL*Plus to Start Up a Database

- Starting an Instance: Scenarios

Options for Starting Up a Database

You can start up and administer an instance of your database if several says, as described in the sections that follow.

Starting Up a Database Using SQL*Plus

You can start a SQL*Plus session, connect to Oracle Database with administrator privileges, and then issue the STARTUP command. Using SQL*Plus in this way is the only method described in detail in this book.

Starting Up a Database Using Recovery Manager

You can also use Recovery Manager (RMAN) to execute STARTUP and SHUTDOWN commands. You may prefer to do this if your are within the RMAN environment and do not want to invoke SQL*Plus.

> **See Also:** *Oracle Database Backup and Recovery Basics* for information on starting up the database using RMAN

Starting Up a Database Using Oracle Enterprise Manager

You can use Oracle Enterprise Manager (EM) to administer your database, including starting it up and shutting it down. Enterprise Manager is a separate Oracle product that combines a GUI console, agents, common services, and tools to provide an integrated and comprehensive systems management platform for managing Oracle products. EM enables you to perform the functions discussed in this book using a GUI interface, rather than command line operations.

> **See Also:**
>
> - *Oracle Enterprise Manager Concepts*
> - *Oracle Enterprise Manager Grid Control Installation and Basic Configuration*

The remainder of this section describes using SQL*Plus to start up a database instance.

Preparing to Start an Instance

You must perform some preliminary steps before attempting to start an instance of your database using SQL*Plus.

1. Start SQL*Plus without connecting to the database:

   ```
   SQLPLUS /NOLOG
   ```

2. Connect to Oracle Database as SYSDBA:

   ```
   CONNECT username/password AS SYSDBA
   ```

Now you are connected to the database and ready to start up an instance of your database.

> **See Also:** *SQL*Plus User's Guide and Reference* for descriptions and syntax for the CONNECT, STARTUP, and SHUTDOWN commands. These commands are SQL*Plus commands.

Using SQL*Plus to Start Up a Database

You use the SQL*Plus STARTUP command to start up an Oracle Database instance. To start an instance, the database must read instance configuration parameters (the initialization parameters) from either a server parameter file or a traditional text initialization parameter file.

When you issue the STARTUP command, by default, the database reads the initialization parameters from a server parameter file (SPFILE) in a platform-specific default location. If you have not created a server parameter file, or if you wish to use a traditional text parameter file instead, you must specify the PFILE clause of the STARTUP command to identify the initialization parameter file.

Note: For UNIX, the platform-specific default location (directory) for the server parameter file (or text initialization parameter file) is:

```
$ORACLE_HOME/dbs
```

For Windows NT and Windows 2000 the location is:

```
%ORACLE_HOME%\database
```

In the platform-specific default location, Oracle Database locates your initialization parameter file by examining filenames in the following order:

1. spfile*$ORACLE_SID*.ora
2. spfile.ora
3. init*$ORACLE_SID*.ora

> **Note:** The spfile.ora file is included in this search path because in a Real Application Clusters environment one server parameter file is used to store the initialization parameter settings for all instances. There is no instance-specific location for storing a server parameter file.
>
> For more information about the server parameter file for a Real Application Clusters environment, see *Oracle Real Application Clusters Administrator's Guide.*

You can direct the database to read initialization parameters from a traditional text initialization parameter file, by using the PFILE clause of the STARTUP command. For example:

```
STARTUP PFILE = /u01/oracle/dbs/init.ora
```

It is not usually necessary to start an instance with a nondefault server parameter file. However, should such a need arise, you can use this PFILE clause to start an instance with a nondefault server parameter file as follows:

1. Create a one-line text initialization parameter file that contains only the SPFILE parameter. The value of the parameter is the nondefault server parameter file location.

   For example, create a text initialization parameter file /u01/oracle/dbs/spf_init.ora that contains only the following parameter:

   ```
   SPFILE = /u01/oracle/dbs/test_spfile.ora
   ```

   > **Note:** You cannot use the IFILE initialization parameter within a text initialization parameter file to point to a server parameter file. In this context, you must use the SPFILE initialization parameter.

2. Start up the instance pointing to this initialization parameter file.

   ```
   STARTUP PFILE = /u01/oracle/dbs/spf_init.ora
   ```

The server parameter file must reside on the machine running the database server. Therefore, the preceding method also provides a means for a client machine to start a database that uses a server parameter file. It also eliminates the need for a client machine to maintain a client-side initialization parameter file. When the client machine reads the initialization parameter file containing the SPFILE parameter, it passes the value to the server where the specified server parameter file is read.

You can start an instance in various modes:

- Start the instance without mounting a database. This does not allow access to the database and usually would be done only for database creation or the re-creation of control files.
- Start the instance and mount the database, but leave it closed. This state allows for certain DBA activities, but does not allow general access to the database.
- Start the instance, and mount and open the database. This can be done in unrestricted mode, allowing access to all users, or in restricted mode, allowing access for database administrators only.

Note: You cannot start a database instance if you are connected to the database through a shared server process.

In addition, you can force the instance to start, or start the instance and have complete media recovery begin immediately. The STARTUP command clauses that you specify to achieve these states are illustrated in the following section.

> **See Also:** Chapter 2, "Creating an Oracle Database", for more information about initialization parameters, initialization parameter files, and server parameter files

Starting an Instance: Scenarios

The following scenarios describe and illustrate the various states in which you can start up an instance. Some restrictions apply when combining clauses of the STARTUP command.

Note: It is possible to encounter problems starting up an instance if control files, database files, or redo log files are not available. If one or more of the files specified by the CONTROL_FILES initialization parameter does not exist or cannot be opened when you attempt to mount a database, Oracle Database returns a warning message and does not mount the database. If one or more of the datafiles or redo log files is not available or cannot be opened when attempting to open a database, the database returns a warning message and does not open the database.

See Also: *SQL*Plus User's Guide and Reference* for information about the restrictions that apply when combining clauses of the STARTUP command

Starting an Instance, and Mounting and Opening a Database

Normal database operation means that an instance is started and the database is mounted and open. This mode allows any valid user to connect to the database and perform typical data access operations.

Start an instance, read the initialization parameters from the default server parameter file location, and then mount and open the database by using the STARTUP command by itself (you can, of course, optionally specify the PFILE clause):

```
STARTUP
```

Starting an Instance Without Mounting a Database

You can start an instance without mounting a database. Typically, you do so only during database creation. Use the STARTUP command with the NOMOUNT clause:

```
STARTUP NOMOUNT
```

Starting an Instance and Mounting a Database

You can start an instance and mount a database without opening it, allowing you to perform specific maintenance operations. For example, the database must be mounted but not open during the following tasks:

- Enabling and disabling redo log archiving options. For more information, please refer to Chapter 7, "Managing Archived Redo Logs".

- Performing full database recovery. For more information, please refer to *Oracle Database Backup and Recovery Basics*

Start an instance and mount the database, but leave it closed by using the STARTUP command with the MOUNT clause:

```
STARTUP MOUNT
```

Restricting Access to an Instance at Startup

You can start an instance, and optionally mount and open a database, in restricted mode so that the instance is available only to administrative personnel (not general database users). Use this mode of instance startup when you need to accomplish one of the following tasks:

- Perform an export or import of database data
- Perform a data load (with SQL*Loader)
- Temporarily prevent typical users from using data
- During certain migration and upgrade operations

Typically, all users with the CREATE SESSION system privilege can connect to an open database. Opening a database in restricted mode allows database access only to users with both the CREATE SESSION and RESTRICTED SESSION system privilege. Only database administrators should have the RESTRICTED SESSION system privilege. Further, when the instance is in restricted mode, a database administrator cannot access the instance remotely through an Oracle Net listener, but can only access the instance locally from the machine that the instance is running on.

Start an instance (and, optionally, mount and open the database) in restricted mode by using the STARTUP command with the RESTRICT clause:

```
STARTUP RESTRICT
```

Later, use the ALTER SYSTEM statement to disable the RESTRICTED SESSION feature:

```
ALTER SYSTEM DISABLE RESTRICTED SESSION;
```

If you open the database in nonrestricted mode and later find you need to restrict access, you can use the ALTER SYSTEM statement to do so, as described in "Restricting Access to an Open Database" on page 3-10.

> **See Also:** *Oracle Database SQL Reference* for more information on the ALTER SYSTEM statement

Forcing an Instance to Start

In unusual circumstances, you might experience problems when attempting to start a database instance. You should not force a database to start unless you are faced with the following:

- You cannot shut down the current instance with the SHUTDOWN NORMAL, SHUTDOWN IMMEDIATE, or SHUTDOWN TRANSACTIONAL commands.
- You experience problems when starting an instance.

If one of these situations arises, you can usually solve the problem by starting a new instance (and optionally mounting and opening the database) using the STARTUP command with the FORCE clause:

```
STARTUP FORCE
```

If an instance is running, STARTUP FORCE shuts it down with mode ABORT before restarting it.

> **See Also:** "Shutting Down with the ABORT Clause" on page 3-13 to understand the side effects of aborting the current instance

Starting an Instance, Mounting a Database, and Starting Complete Media Recovery

If you know that media recovery is required, you can start an instance, mount a database to the instance, and have the recovery process automatically start by using the STARTUP command with the RECOVER clause:

```
STARTUP OPEN RECOVER
```

If you attempt to perform recovery when no recovery is required, Oracle Database issues an error message.

Automatic Database Startup at Operating System Start

Many sites use procedures to enable automatic startup of one or more Oracle Database instances and databases immediately following a system start. The procedures for performing this task are specific to each operating system. For information about automatic startup, see your operating system specific Oracle documentation.

Starting Remote Instances

If your local Oracle Database server is part of a distributed database, you might want to start a remote instance and database. Procedures for starting and stopping remote instances vary widely depending on communication protocol and operating system.

Altering Database Availability

You can alter the availability of a database. You may want to do this in order to restrict access for maintenance reasons or to make the database read only. The following sections explain how to alter the availability of a database:

- Mounting a Database to an Instance
- Opening a Closed Database
- Opening a Database in Read-Only Mode
- Restricting Access to an Open Database

Mounting a Database to an Instance

When you need to perform specific administrative operations, the database must be started and mounted to an instance, but closed. You can achieve this scenario by starting the instance and mounting the database.

To mount a database to a previously started, but not opened instance, use the SQL statement `ALTER DATABASE` with the `MOUNT` clause as follows:

```
ALTER DATABASE MOUNT;
```

> **See Also:** "Starting an Instance and Mounting a Database" on page 3-6 for a list of operations that require the database to be mounted and closed (and procedures to start an instance and mount a database in one step)

Opening a Closed Database

You can make a mounted but closed database available for general use by opening the database. To open a mounted database, use the `ALTER DATABASE` statement with the `OPEN` clause:

```
ALTER DATABASE OPEN;
```

After executing this statement, any valid Oracle Database user with the CREATE SESSION system privilege can connect to the database.

Opening a Database in Read-Only Mode

Opening a database in read-only mode enables you to query an open database while eliminating any potential for online data content changes. While opening a database in read-only mode guarantees that datafile and redo log files are not written to, it does not restrict database recovery or operations that change the state of the database without generating redo. For example, you can take datafiles offline or bring them online since these operations do not affect data content.

If a query against a database in read-only mode uses temporary tablespace, for example to do disk sorts, then the issuer of the query must have a locally managed tablespace assigned as the default temporary tablespace. Otherwise, the query will fail. This is explained in "Creating a Locally Managed Temporary Tablespace" on page 8-18.

Ideally, you open a database in read-only mode when you alternate a standby database between read-only and recovery mode. Be aware that these are mutually exclusive modes.

The following statement opens a database in read-only mode:

```
ALTER DATABASE OPEN READ ONLY;
```

You can also open a database in read / write mode as follows:

```
ALTER DATABASE OPEN READ WRITE;
```

However, read / write is the default mode.

Note: You cannot use the RESETLOGS clause with a READ ONLY clause.

See Also: *Oracle Database SQL Reference* for more information about the ALTER DATABASE statement

Restricting Access to an Open Database

To place an instance in restricted mode, where only users with administrative privileges can access it, use the SQL statement ALTER SYSTEM with the ENABLE RESTRICTED SESSION clause. After placing an instance in restricted mode, you

should consider killing all current user sessions before performing any administrative tasks.

To lift an instance from restricted mode, use `ALTER SYSTEM` with the `DISABLE RESTRICTED SESSION` clause.

See Also:

- "Terminating Sessions" on page 4-23 for directions for killing user sessions
- "Restricting Access to an Instance at Startup" on page 3-7 to learn some reasons for placing an instance in restricted mode

Shutting Down a Database

To initiate database shutdown, use the SQL*Plus `SHUTDOWN` command. Control is not returned to the session that initiates a database shutdown until shutdown is complete. Users who attempt connections while a shutdown is in progress receive a message like the following:

```
ORA-01090: shutdown in progress - connection is not permitted
```

Note: You cannot shut down a database if you are connected to the database through a shared server process.

To shut down a database and instance, you must first connect as `SYSOPER` or `SYSDBA`. There are several modes for shutting down a database. These are discussed in the following sections:

- Shutting Down with the NORMAL Clause
- Shutting Down with the IMMEDIATE Clause
- Shutting Down with the TRANSACTIONAL Clause
- Shutting Down with the ABORT Clause

Shutting Down with the NORMAL Clause

To shut down a database in normal situations, use the `SHUTDOWN` command with the `NORMAL` clause:

```
SHUTDOWN NORMAL
```

Normal database shutdown proceeds with the following conditions:

- No new connections are allowed after the statement is issued.
- Before the database is shut down, the database waits for all currently connected users to disconnect from the database.

The next startup of the database will not require any instance recovery procedures.

Shutting Down with the IMMEDIATE Clause

Use immediate database shutdown only in the following situations:

- To initiate an automated and unattended backup
- When a power shutdown is going to occur soon
- When the database or one of its applications is functioning irregularly and you cannot contact users to ask them to log off or they are unable to log off

To shut down a database immediately, use the `SHUTDOWN` command with the `IMMEDIATE` clause:

```
SHUTDOWN IMMEDIATE
```

Immediate database shutdown proceeds with the following conditions:

- No new connections are allowed, nor are new transactions allowed to be started, after the statement is issued.
- Any uncommitted transactions are rolled back. (If long uncommitted transactions exist, this method of shutdown might not complete quickly, despite its name.)
- Oracle Database does not wait for users currently connected to the database to disconnect. The database implicitly rolls back active transactions and disconnects all connected users.

The next startup of the database will not require any instance recovery procedures.

Shutting Down with the TRANSACTIONAL Clause

When you want to perform a planned shutdown of an instance while allowing active transactions to complete first, use the `SHUTDOWN` command with the `TRANSACTIONAL` clause:

```
SHUTDOWN TRANSACTIONAL
```

Transactional database shutdown proceeds with the following conditions:

- No new connections are allowed, nor are new transactions allowed to be started, after the statement is issued.
- After all transactions have completed, any client still connected to the instance is disconnected.
- At this point, the instance shuts down just as it would when a `SHUTDOWN IMMEDIATE` statement is submitted.

The next startup of the database will not require any instance recovery procedures.

A transactional shutdown prevents clients from losing work, and at the same time, does not require all users to log off.

Shutting Down with the ABORT Clause

You can shut down a database instantaneously by aborting the database instance. If possible, perform this type of shutdown *only* in the following situations:

The database or one of its applications is functioning irregularly *and* none of the other types of shutdown works.

- You need to shut down the database instantaneously (for example, if you know a power shutdown is going to occur in one minute).
- You experience problems when starting a database instance.

When you must do a database shutdown by aborting transactions and user connections, issue the `SHUTDOWN` command with the `ABORT` clause:

```
SHUTDOWN ABORT
```

An aborted database shutdown proceeds with the following conditions:

- No new connections are allowed, nor are new transactions allowed to be started, after the statement is issued.
- Current client SQL statements being processed by Oracle Database are immediately terminated.
- Uncommitted transactions are not rolled back.
- Oracle Database does not wait for users currently connected to the database to disconnect. The database implicitly disconnects all connected users.

The next startup of the database *will* require instance recovery procedures.

Quiescing a Database

Occasionally you might want to put a database in a state that allows only DBA transactions, queries, fetches, or PL/SQL statements. Such a state is referred to as a **quiesced state**, in the sense that no ongoing non-DBA transactions, queries, fetches, or PL/SQL statements are running in the system.

> **Note:** In this discussion of quiesce database, a DBA is defined as user SYS or SYSTEM. Other users, including those with the DBA role, are not allowed to issue the `ALTER SYSTEM QUIESCE DATABASE` statement or proceed after the database is quiesced.

The quiesced state lets administrators perform actions that cannot safely be done otherwise. These actions include:

- Actions that fail if concurrent user transactions access the same object--for example, changing the schema of a database table or adding a column to an existing table where a no-wait lock is required.
- Actions whose undesirable intermediate effect can be seen by concurrent user transactions--for example, a multistep procedure for reorganizing a table when the table is first exported, then dropped, and finally imported. A concurrent user who attempts to access the table after it was dropped, but before import, would not have an accurate view of the situation.

Without the ability to quiesce the database, you would need to shut down the database and reopen it in restricted mode. This is a serious restriction, especially for systems requiring 24 x 7 availability. Quiescing a database is much a smaller restriction, because it eliminates the disruption to users and the downtime associated with shutting down and restarting the database.

To be able to quiesce the database, you must have the Database Resource Manager feature activated, and it must have been activated since the current instance (or all instances in an Oracle Real Application Clusters environment) started up. It is through the facilities of the Database Resource Manager that non-DBA sessions are prevented from becoming active. Also, while this statement is in effect, any attempt to change the current resource plan will be queued until after the system is unquiesced. Please refer to Chapter 24, "Using the Database Resource Manager" for more information about the Database Resource Manager.

Placing a Database into a Quiesced State

To place a database into a quiesced state, issue the following statement:

```
ALTER SYSTEM QUIESCE RESTRICTED;
```

Non-DBA active sessions will continue until they become inactive. An active session is one that is currently inside of a transaction, a query, a fetch, or a PL/SQL statement; or a session that is currently holding any shared resources (for example, enqueues). No inactive sessions are allowed to become active. For example, If a user issues a SQL query in an attempt to force an inactive session to become active, the query will appear to be hung. When the database is later unquiesced, the session is resumed, and the blocked action is processed.

Once all non-DBA sessions become inactive, the `ALTER SYSTEM QUIESCE RESTRICTED` statement completes, and the database is in a quiesced state. In an Oracle Real Application Clusters environment, this statement affects all instances, not just the one that issues the statement.

The `ALTER SYSTEM QUIESCE RESTRICTED` statement may wait a long time for active sessions to become inactive. If you interrupt the request, or if your session terminates abnormally before all active sessions are quiesced, then Oracle Database will automatically reverses any partial effects of the statement.

For queries that are carried out by successive multiple Oracle Call Interface (OCI) fetches, the `ALTER SYSTEM QUIESCE RESTRICTED` statement does not wait for all fetches to finish. It only waits for the current fetch to finish.

For both dedicated and shared server connections, all non-DBA logins after this statement is issued are queued by the Database Resource Manager, and are not allowed to proceed. To the user, it appears as if the login is hung. The login will resume when the database is unquiesced.

The database remains in the quiesced state even if the session that issued the statement exits. A DBA must log in to the database to issue the statement that specifically unquiesces the database.

Note: You cannot perform a cold backup when the database is in the quiesced state, because Oracle Database background processes may still perform updates for internal purposes even while the database is quiesced. In addition, the file headers of online datafiles continue to appear to be accessible. They do not look the same as if a clean shutdown had been performed. However, you can still take online backups while the database is in a quiesced state.

Restoring the System to Normal Operation

The following statement restores the database to normal operation:

```
ALTER SYSTEM UNQUIESCE;
```

All non-DBA activity is allowed to proceed. In an Oracle Real Application Clusters environment, this statement is not required to be issued from the same session, or even the same instance, as that which quiesced the database. If the session issuing the ALTER SYSTEM UNQUIESCE statement terminates abnormally, then the Oracle Database server ensures that the unquiesce operation completes.

Viewing the Quiesce State of an Instance

You can query the ACTIVE_STATE column of the V$INSTANCE view to see the current state of an instance. The column values has one of these values:

- NORMAL: Normal unquiesed state.
- QUIESCING: Being quiesced, but some non-DBA sessions are still active.
- QUIESCED: Quiesced; no non-DBA sessions are active or allowed.

Suspending and Resuming a Database

The ALTER SYSTEM SUSPEND statement halts all input and output (I/O) to datafiles (file header and file data) and control files. The suspended state lets you back up a database without I/O interference. When the database is suspended all preexisting I/O operations are allowed to complete and any new database accesses are placed in a queued state.

The suspend command is not specific to an instance. In an Oracle Real Application Clusters environment, when you issue the suspend command on one system, internal locking mechanisms propagate the halt request across instances, thereby quiescing all active instances in a given cluster. However, if someone starts a new instance another instance is being suspended, the new instance will not be suspended.

Use the ALTER SYSTEM RESUME statement to resume normal database operations. The SUSPEND and RESUME commands can be issued from different instances. For example, if instances 1, 2, and 3 are running, and you issue an ALTER SYSTEM SUSPEND statement from instance 1, then you can issue a RESUME statement from instance 1, 2, or 3 with the same effect.

The suspend / resume feature is useful in systems that allow you to mirror a disk or file and then split the mirror, providing an alternative backup and restore solution. If you use a system that is unable to split a mirrored disk from an existing database while writes are occurring, then you can use the suspend / resume feature to facilitate the split.

The suspend / resume feature is not a suitable substitute for normal shutdown operations, because copies of a suspended database can contain uncommitted updates.

Caution: Do not use the `ALTER SYSTEM SUSPEND` statement as a substitute for placing a tablespace in hot backup mode. Precede any database suspend operation by an `ALTER TABLESPACE BEGIN BACKUP` statement.

The following statements illustrate `ALTER SYSTEM SUSPEND/RESUME` usage. The `V$INSTANCE` view is queried to confirm database status.

```
SQL> ALTER SYSTEM SUSPEND;
System altered
SQL> SELECT DATABASE_STATUS FROM V$INSTANCE;
DATABASE_STATUS
---------
SUSPENDED

SQL> ALTER SYSTEM RESUME;
System altered
SQL> SELECT DATABASE_STATUS FROM V$INSTANCE;
DATABASE_STATUS
---------
ACTIVE
```

See Also: *Oracle Database Backup and Recovery Advanced User's Guide* for details about backing up a database using the database suspend / resume feature

4

Managing Oracle Database Processes

This chapter describes how to manage and monitor the processes of an Oracle Database instance and contains the following topics:

- About Dedicated and Shared Server Processes
- Configuring Oracle Database for Shared Server
- About Oracle Database Background Processes
- Managing Processes for Parallel SQL Execution
- Managing Processes for External Procedures
- Terminating Sessions
- Monitoring the Operation of Your Database

About Dedicated and Shared Server Processes

Oracle Database creates server processes to handle the requests of user processes connected to an instance. A server process can be either of the following:

- A **dedicated server process**, which services only one user process
- A **shared server process**, which can service multiple user processes

Your database is always enabled to allow dedicated server processes, but you must specifically configure and enable **shared server** by setting one or more initialization parameters.

Dedicated Server Processes

Figure 4–1, "Oracle Database Dedicated Server Processes" illustrates how dedicated server processes work. In this diagram two user processes are connected to the database through dedicated server processes.

In general, it is better to be connected through a **dispatcher** and use a shared server process. This is illustrated in Figure 4–2, "Oracle Database Shared Server Processes". A shared server process can be more efficient because it keeps the number of processes required for the running instance low.

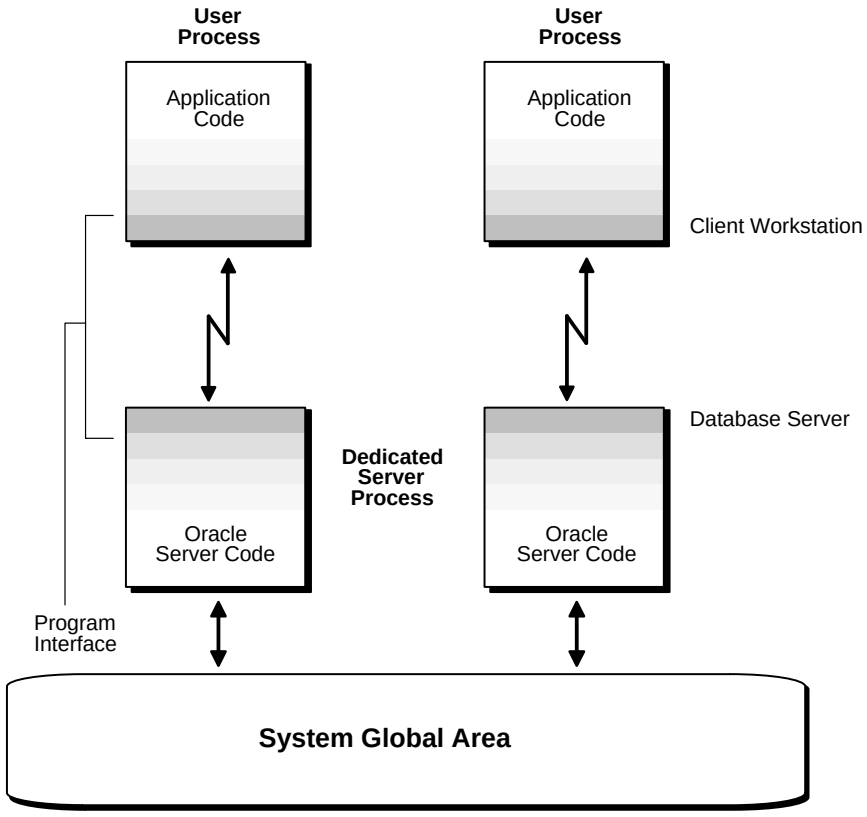
In the following situations, however, users and administrators should explicitly connect to an instance using a dedicated server process:

- To submit a batch job (for example, when a job can allow little or no idle time for the server process)
- To use Recovery Manager (RMAN) to back up, restore, or recover a database

To request a dedicated server connection when Oracle Database is configured for shared server, users must connect using a net service name that is configured to use a dedicated server. Specifically, the net service name value should include the `SERVER=DEDICATED` clause in the connect descriptor.

> **See Also:** *Oracle Net Services Administrator's Guide* for more information about requesting a dedicated server connection

Figure 4–1 Oracle Database Dedicated Server Processes

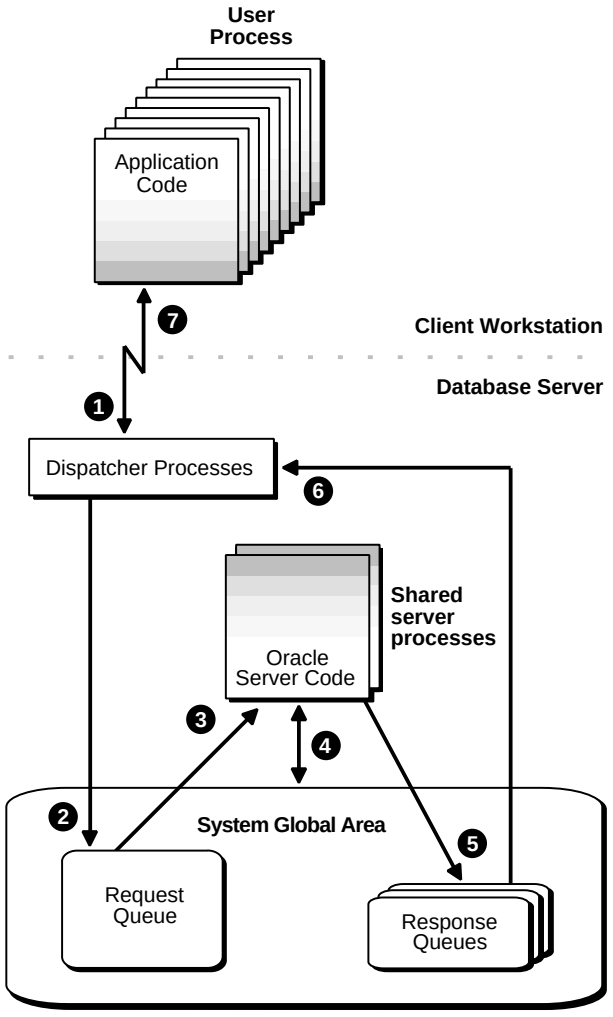


Shared Server Processes

Consider an order entry system with dedicated server processes. A customer phones the order desk and places an order, and the clerk taking the call enters the order into the database. For most of the transaction, the clerk is on the telephone talking to the customer. A server process is not needed during this time, so the server process dedicated to the clerk's user process remains idle. The system is slower for other clerks entering orders, because the idle server process is holding system resources.

Shared server architecture eliminates the need for a dedicated server process for each connection (see Figure 4–2).

Figure 4–2 Oracle Database Shared Server Processes



In a shared server configuration, client user processes connect to a dispatcher. The dispatcher can support multiple client connections concurrently. Each client connection is bound to a **virtual circuit**, which is a piece of shared memory used by the dispatcher for client database connection requests and replies. The dispatcher places a virtual circuit on a common queue when a request arrives.

An idle shared server process picks up the virtual circuit from the common queue, services the request, and relinquishes the virtual circuit before attempting to retrieve another virtual circuit from the common queue. This approach enables a small pool of server processes to serve a large number of clients. A significant advantage of shared server architecture over the dedicated server model is the reduction of system resources, enabling the support of an increased number of users.

For even better resource management, shared server can be configured for **connection pooling**. Connection pooling lets a dispatcher support more users by enabling the database server to time-out protocol connections and to use those connections to service an active session. Further, shared server can be configured for **session multiplexing**, which combines multiple sessions for transmission over a single network connection in order to conserve the operating system's resources.

Shared server architecture requires Oracle Net Services. User processes targeting the shared server must connect through Oracle Net Services, even if they are on the same machine as the Oracle Database instance.

> **See Also:** *Oracle Net Services Administrator's Guide* for more detailed information about shared server, including features such as connection pooling and session multiplexing

Configuring Oracle Database for Shared Server

Shared memory resources are preconfigured to allow the enabling of shared server at run time. You need not configure it by specifying parameters in your initialization parameter file, but you can do so if that better suits your environment. You can start dispatchers and shared server processes (shared servers) dynamically using the `ALTER SYSTEM` statement.

This section discusses how to enable shared server and how to set or alter shared server initialization parameters. It contains the following topics:

- Initialization Parameters for Shared Server
- Enabling Shared Server
- Configuring Dispatchers
- Monitoring Shared Server

> **See Also:** *Oracle Database SQL Reference* for further information about the `ALTER SYSTEM` statement

Initialization Parameters for Shared Server

The following initialization parameters control shared server operation:

- SHARED_SERVERS: Specifies the initial number of shared servers to start and the minimum number of shared servers to keep. This is the only required parameter for using shared servers.
- MAX_SHARED_SERVERS: Specifies the maximum number of shared servers that can run simultaneously.
- SHARED_SERVER_SESSIONS: Specifies the total number of shared server user sessions that can run simultaneously. Setting this parameter enables you to reserve user sessions for dedicated servers.
- DISPATCHERS: Configures dispatcher processes in the shared server architecture.
- MAX_DISPATCHERS: Specifies the maximum number of dispatcher processes that can run simultaneously. This parameter can be ignored for now. It will only be useful in a future release when the number of dispatchers is auto-tuned according to the number of concurrent connections.
- CIRCUITS: Specifies the total number of virtual circuits that are available for inbound and outbound network sessions.

> **See Also:** *Oracle Database Reference* for more information about these initialization parameters

Enabling Shared Server

Shared server is enabled by setting the SHARED_SERVERS initialization parameter to a value greater than 0. The other shared server initialization parameters need not be set. Because shared server requires at least one dispatcher in order to work, a dispatcher is brought up even if no dispatcher has been configured. Dispatchers are discussed in "Configuring Dispatchers" on page 4-9.

Shared server can be started dynamically by setting the SHARED_SERVERS parameter to a nonzero value with the ALTER SYSTEM statement, or SHARED_SERVERS can be included at database startup in the initialization parameter file. If SHARED_SERVERS is not included in the initialization parameter file, or is included but is set to 0, then shared server is not enabled at database startup.

Note: For backward compatibility, if SHARED_SERVERS is not included in the initialization parameter file at database startup, but DISPATCHERS is included and it specifies at least one dispatcher, shared server is enabled. In this case, the default for SHARED_SERVERS is 1.

However, if neither SHARED_SERVERS nor DISPATCHERS is included in the initialization file, you cannot start shared server after the instance is brought up by just altering the DISPATCHERS parameter. You must specifically alter SHARED_SERVERS to a nonzero value to start shared server.

Determining a Value for SHARED_SERVERS

The SHARED_SERVERS initialization parameter specifies the minimum number of shared servers that you want created when the instance is started. After instance startup, Oracle Database can dynamically adjust the number of shared servers based on how busy existing shared servers are and the length of the request queue.

In typical systems, the number of shared servers stabilizes at a ratio of one shared server for every ten connections. For OLTP applications, when the rate of requests is low, or when the ratio of server usage to request is low, the connections-to-servers ratio could be higher. In contrast, in applications where the rate of requests is high or the server usage-to-request ratio is high, the connections-to-server ratio could be lower.

The PMON (process monitor) background process cannot terminate shared servers below the value specified by SHARED_SERVERS. Therefore, you can use this parameter to stabilize the load and minimize strain on the system by preventing PMON from terminating and then restarting shared servers because of coincidental fluctuations in load.

If you know the average load on your system, you can set SHARED_SERVERS to an optimal value. The following example shows how you can use this parameter:

Assume a database is being used by a telemarketing center staffed by 1000 agents. On average, each agent spends 90% of the time talking to customers and only 10% of the time looking up and updating records. To keep the shared servers from being terminated as agents talk to customers and then spawned again as agents access the database, a DBA specifies that the optimal number of shared servers is 100.

However, not all work shifts are staffed at the same level. On the night shift, only 200 agents are needed. Since SHARED_SERVERS is a dynamic parameter, a DBA

reduces the number of shared servers to 20 at night, thus allowing resources to be freed up for other tasks such as batch jobs.

Decreasing the Number of Shared Server Processes

You can decrease the minimum number of shared servers that must be kept active by dynamically setting the SHARED_SERVERS parameter to a lower value. Thereafter, until the number of shared servers is decreased to the value of the SHARED_SERVERS parameter, any shared servers that become inactive are marked by PMON for termination.

The following statement reduces the number of shared servers:

```
ALTER SYSTEM SET SHARED_SERVERS = 5;
```

Setting SHARED_SERVERS to 0 disables shared server. For more information, please refer to "Disabling Shared Servers" on page 4-16.

Limiting the Number of Shared Server Processes

The MAX_SHARED_SERVERS parameter specifies the maximum number of shared servers that can be automatically created by PMON. It has no default value. If no value is specified, then PMON starts as many shared servers as is required by the load, subject to these limitations:

- The process limit (set by the PROCESSES initialization parameter)
- A minimum number of free process slots (at least one-eighth of the total process slots, or two slots if PROCESSES is set to less than 24)
- System resources

> **Note:** On Windows NT, take care when setting MAX_SHARED_SERVERS to a high value, because each server is a thread in a common process.

The value of SHARED_SERVERS overrides the value of MAX_SHARED_SERVERS. Therefore, you can force PMON to start more shared servers than the MAX_SHARED_SERVERS value by setting SHARED_SERVERS to a value higher than MAX_SHARED_SERVERS. You can subsequently place a new upper limit on the number of shared servers by dynamically altering the MAX_SHARED_SERVERS to a value higher than SHARED_SERVERS.

The primary reason to limit the number of shared servers is to reserve resources, such as memory and CPU time, for other processes. For example, consider the case of the telemarketing center discussed previously:

The DBA wants to reserve two thirds of the resources for batch jobs at night. He sets MAX_SHARED_SERVERS to less than one third of the maximum number of processes (PROCESSES). By doing so, the DBA ensures that even if all agents happen to access the database at the same time, batch jobs can connect to dedicated servers without having to wait for the shared servers to be brought down after processing agents' requests.

Another reason to limit the number of shared servers is to prevent the concurrent run of too many server processes from slowing down the system due to heavy swapping, although PROCESSES can serve as the upper bound for this rather than MAX_SHARED_SERVERS.

Still other reasons to limit the number of shared servers are testing, debugging, performance analysis, and tuning. For example, to see how many shared servers are needed to efficiently support a certain user community, you can vary MAX_SHARED_SERVERS from a very small number upward until no delay in response time is noticed by the users.

Limiting the Number of Shared Server Sessions

The SHARED_SERVER_SESSIONS initialization parameter specifies the maximum number of concurrent shared server user sessions. Setting this parameter, which is a dynamic parameter, lets you reserve database sessions for dedicated servers. This in turn ensures that administrative tasks that require dedicated servers, such as backing up or recovering the database, are not preempted by shared server sessions.

This parameter has no default value. If it is not specified, the system can create shared server sessions as needed, limited by the SESSIONS initialization parameter.

Protecting Shared Memory

The CIRCUITS parameter sets a maximum limit on the number of virtual circuits that can be created in shared memory. This parameter has no default. If it is not specified, then the system can create circuits as needed, limited by the DISPATCHERS initialization parameter and system resources.

Configuring Dispatchers

The DISPATCHERS initialization parameter configures dispatcher processes in the shared server architecture. At least one dispatcher process is required for shared

server to work.If you do not specify a dispatcher, but you enable shared server by setting SHARED_SERVER to a nonzero value, then by default Oracle Database creates one dispatcher for the TCP protocol. The equivalent DISPATCHERS explicit setting of the initialization parameter for this configuration is:

```
dispatchers="(PROTOCOL=tcp)"
```

You can configure more dispatchers, using the DISPATCHERS initialization parameter, if either of the following conditions apply:

- You need to configure a protocol other than TCP/IP. You configure a protocol address with one of the following attributes of the DISPATCHERS parameter:
  - ADDRESS
  - DESCRIPTION
  - PROTOCOL
- You want to configure one or more of the optional dispatcher attributes:
  - DISPATCHERS
  - CONNECTIONS
  - SESSIONS
  - TICKS
  - LISTENER
  - MULTIPLEX
  - POOL
  - SERVICE

Note: Database Configuration Assistant helps you configure this parameter.

DISPATCHERS Initialization Parameter Attributes

This section provides brief descriptions of the attributes that can be specified with the DISPATCHERS initialization parameter.

A protocol address is required and is specified using one or more of the following attributes:

Attribute	Description
ADDRESS	Specify the network protocol address of the endpoint on which the dispatchers listen.
DESCRIPTION	Specify the network description of the endpoint on which the dispatchers listen, including the network protocol address. The syntax is as follows: `(DESCRIPTION=(ADDRESS=...))`
PROTOCOL	Specify the network protocol for which the dispatcher generates a listening endpoint. For example: `(PROTOCOL=tcp)` See the *Oracle Net Services Reference Guide* for further information about protocol address syntax.

The following attribute specifies how many dispatchers this configuration should have. It is optional and defaults to 1.

Attribute	Description
DISPATCHERS	Specify the initial number of dispatchers to start.

The following attributes tell the instance about the network attributes of each dispatcher of this configuration. They are all optional.

Attribute	Description
CONNECTIONS	Specify the maximum number of network connections to allow for each dispatcher.
SESSIONS	Specify the maximum number of network sessions to allow for each dispatcher.
TICKS	Specify the duration of a TICK in seconds. A TICK is a unit of time in terms of which the connection pool timeout can be specified. Used for connection pooling.
LISTENER	Specify an alias name for the listeners with which the PMON process registers dispatcher information. Set the alias to a name that is resolved through a naming method.
MULTIPLEX	Used to enable the Oracle Connection Manager session multiplexing feature.
POOL	Used to enable connection pooling.

Attribute	Description
SERVICE	Specify the service names the dispatchers register with the listeners.

You can specify either an entire attribute name a substring consisting of at least the first three characters. For example, you can specify SESSIONS=3, SES=3, SESS=3, or SESSI=3, and so forth.

> **See Also:** *Oracle Database Reference* for more detailed descriptions of the attributes of the DISPATCHERS initialization parameter

Determining the Number of Dispatchers

Once you know the number of possible connections for each process for the operating system, calculate the initial number of dispatchers to create during instance startup, for each network protocol, using the following formula:

```
Number of dispatchers =
   CEIL ( max. concurrent sessions / connections for each dispatcher )
```

CEIL returns the result roundest up to the next whole integer.

For example, assume a system that can support 970 connections for each process, and that has:

- A maximum of 4000 sessions concurrently connected through TCP/IP and
- A maximum of 2,500 sessions concurrently connected through TCP/IP with SSL

The DISPATCHERS attribute for TCP/IP should be set to a minimum of five dispatchers (4000 / 970), and for TCP/IP with SSL three dispatchers (2500 / 970:

```
DISPATCHERS='(PROT=tcp)(DISP=5)', '(PROT-tcps)(DISP=3)'
```

Depending on performance, you may need to adjust the number of dispatchers.

Setting the Initial Number of Dispatchers

You can specify multiple dispatcher configurations by setting DISPATCHERS to a comma separated list of strings, or by specifying multiple DISPATCHERS parameters in the initialization file. If you specify DISPATCHERS multiple times, the lines must be adjacent to each other in the initialization parameter file. Internally, Oracle Database assigns an INDEX value (beginning with zero) to each

DISPATCHERS parameter. You can later refer to that DISPATCHERS parameter in an ALTER SYSTEM statement by its index number.

Some examples of setting the DISPATCHERS initialization parameter follow.

Example: Typical This is a typical example of setting the DISPATCHERS initialization parameter.

```
DISPATCHERS="(PROTOCOL=TCP)(DISPATCHERS=2)"
```

Example: Forcing the IP Address Used for Dispatchers The following hypothetical example will create two dispatchers that will listen on the specified IP address. The address must be a valid IP address for the host that the instance is on. (The host may be configured with multiple IP addresses.)

```
DISPATCHERS="(ADDRESS=(PROTOCOL=TCP)(HOST=144.25.16.201))(DISPATCHERS=2)"
```

Example: Forcing the Port Used by Dispatchers To force the dispatchers to use a specific port as the listening endpoint, add the PORT attribute as follows:

```
DISPATCHERS="(ADDRESS=(PROTOCOL=TCP)(PORT=5000))"
DISPATCHERS="(ADDRESS=(PROTOCOL=TCP)(PORT=5001))"
```

Altering the Number of Dispatchers

You can control the number of dispatcher processes in the instance. Unlike the number of shared servers, the number of dispatchers does not change automatically. You change the number of dispatchers explicitly with the ALTER SYSTEM statement. In this release of Oracle Database, you can increase the number of dispatchers to more than the limit specified by the MAX_DISPATCHERS parameter. It is planned that MAX_DISPATCHERS will be taken into consideration in a future release.

Monitor the following views to determine the load on the dispatcher processes:

- V$QUEUE
- V$DISPATCHER
- V$DISPATCHER_RATE

> **See Also:** *Oracle Database Performance Tuning Guide* for information about monitoring these views to determine dispatcher load and performance

If these views indicate that the load on the dispatcher processes is consistently high, then performance may be improved by starting additional dispatcher processes to route user requests. In contrast, if the load on dispatchers is consistently low, reducing the number of dispatchers may improve performance.

To dynamically alter the number of dispatchers when the instance is running, use the ALTER SYSTEM statement to modify the DISPATCHERS attribute setting for an existing dispatcher configuration. You can also add new dispatcher configurations to start dispatchers with different network attributes.

When you reduce the number of dispatchers for a particular dispatcher configuration, the dispatchers are not immediately removed. Rather, as users disconnect, Oracle Database terminates dispatchers down to the limit you specify in DISPATCHERS,

For example, suppose the instance was started with this DISPATCHERS setting in the initialization parameter file:

```
DISPATCHERS='(PROT=tcp)(DISP=2)', '(PROT=tcps)(DISP=2)'
```

To increase the number of dispatchers for the TCP/IP protocol from 2 to 3, and decrease the number of dispatchers for the TCP/IP with SSL protocol from 2 to 1, you can issue the following statement:

```
ALTER SYSTEM SET DISPATCHERS = '(INDEX=0)(DISP=3)', '(INDEX=1)(DISP=1)';
```

or

```
ALTER SYSTEM SET DISPATCHERS = '(PROT=tcp)(DISP=3)', '(PROT-tcps)(DISP=1)';
```

Note: You need not specify (DISP=1). It is optional because 1 is the default value for the DISPATCHERS parameter.

If fewer than three dispatcher processes currently exist for TCP/IP, the database creates new ones. If more than one dispatcher process currently exists for TCP/IP with SSL, then the database terminates the extra ones as the connected users disconnect.

Suppose that instead of changing the number of dispatcher processes for the TCP/IP protocol, you want to add another TCP/IP dispatcher that supports connection pooling. You can do so by entering the following statement:

```
ALTER SYSTEM SET DISPATCHERS = '(INDEX=2)(PROT=tcp)(POOL=on)';
```

The INDEX attribute is needed to add the new dispatcher configuration. If you omit (INDEX=2) in the preceding statement, then the TCP/IP dispatcher configuration at INDEX 0 will be changed to support connection pooling, and the number of dispatchers for that configuration will be reduced to 1, which is the default when the number of dispatchers (attribute DISPATCHERS) is not specified.

Notes on Altering Dispatchers

- The INDEX keyword can be used to identify which dispatcher configuration to modify. If you do not specify INDEX, then the first dispatcher configuration matching the DESCRIPTION, ADDRESS, or PROTOCOL specified will be modified. If no match is found among the existing dispatcher configurations, then a new dispatcher will be added.
- The INDEX value can range from 0 to *n*-1, where *n* is the current number of dispatcher configurations. If your ALTER SYSTEM statement specifies an INDEX value equal to *n*, where *n* is the current number of dispatcher configurations, a new dispatcher configuration will be added.
- To see the values of the current dispatcher configurations--that is, the number of dispatchers, whether connection pooling is on, and so forth--query the V$DISPATCHER_CONFIG dynamic performance view. To see which dispatcher configuration a dispatcher is associated with, query the CONF_INDX column of the V$DISPATCHER view.
- When you change the DESCRIPTION, ADDRESS, PROTOCOL, CONNECTIONS, TICKS, MULTIPLEX, and POOL attributes of a dispatcher configuration, the change does not take effect for existing dispatchers but only for new dispatchers. Therefore, in order for the change to be effective for all dispatchers associated with a configuration, you must forcibly kill existing dispatchers after altering the DISPATCHERS parameter, and let the database start new ones in their place with the newly specified properties.

  The attributes LISTENER and SERVICES are not subject to the same constraint. They apply to existing dispatchers associated with the modified configuration. Attribute SESSIONS applies to existing dispatchers only if its value is reduced. However, if its value is increased, it is applied only to newly started dispatchers.

Shutting Down Specific Dispatcher Processes

With the ALTER SYSTEM statement, you leave it up to the database to determine which dispatchers to shut down to reduce the number of dispatchers. Alternatively, it is possible to shut down specific dispatcher processes. To identify the name of the

specific dispatcher process to shut down, use the V$DISPATCHER dynamic performance view.

```
SELECT NAME, NETWORK FROM V$DISPATCHER;
```

Each dispatcher is uniquely identified by a name of the form D*nnn*.

To shut down dispatcher D002, issue the following statement:

```
ALTER SYSTEM SHUTDOWN IMMEDIATE 'D002';
```

The IMMEDIATE keyword stops the dispatcher from accepting new connections and the database immediately terminates all existing connections through that dispatcher. After all sessions are cleaned up, the dispatcher process shuts down. If IMMEDIATE were not specified, the dispatcher would wait until all of its users disconnected and all of its connections terminated before shutting down.

Disabling Shared Servers

You disable shared server by setting SHARED_SERVERS to 0. No new client can connect in shared mode. However, when you set SHARED_SERVERS to 0, Oracle Database retains some shared servers until all shared server connections are closed. The number of shared servers retained is either the number specified by the preceding setting of SHARED_SERVERS or the value of the MAX_SHARED_SERVERS parameter, whichever is smaller. If both SHARED_SERVERS and MAX_SHARED_SERVERS are set to 0, then all shared servers will terminate and requests from remaining shared server clients will be queued until the value of SHARED_SERVERS or MAX_SHARED_SERVERS is raised again.

To terminate dispatchers once all shared server clients disconnect, enter this statement:

```
ALTER SYSTEM SET DISPATCHERS = '';
```

Monitoring Shared Server

The following views are useful for obtaining information about your shared server configuration and for monitoring performance.

View	Description
V$DISPATCHER	Provides information on the dispatcher processes, including name, network address, status, various usage statistics, and index number.

View	Description
V$DISPATCHER_CONFIG	Provides configuration information about the dispatchers.
V$DISPATCHER_RATE	Provides rate statistics for the dispatcher processes.
V$QUEUE	Contains information on the shared server message queues.
V$SHARED_SERVER	Contains information on the shared servers.
V$CIRCUIT	Contains information about virtual circuits, which are user connections to the database through dispatchers and servers.
V$SHARED_SERVER_MONITOR	Contains information for tuning shared server.
V$SGA	Contains size information about various system global area (SGA) groups. May be useful when tuning shared server.
V$SGASTAT	Contains detailed statistical information about the SGA, useful for tuning.
V$SHARED_POOL_RESERVED	Lists statistics to help tune the reserved pool and space within the shared pool.

See Also:

- *Oracle Database Reference* for detailed descriptions of these views
- *Oracle Database Performance Tuning Guide* for specific information about monitoring and tuning shared server

About Oracle Database Background Processes

To maximize performance and accommodate many users, a multiprocess Oracle Database system uses **background processes**. Background processes consolidate functions that would otherwise be handled by multiple database programs running for each user process. Background processes asynchronously perform I/O and monitor other Oracle Database processes to provide increased parallelism for better performance and reliability.

Table 4–1 describes the basic Oracle Database background processes, many of which are discussed in more detail elsewhere in this book. The use of additional database server features or options can cause more background processes to be present. For

example, when you use Advanced Queuing, the queue monitor (QMN*n*) background process is present. When you specify the `FILE_MAPPING` initialization parameter for mapping datafiles to physical devices on a storage subsystem, then the FMON process is present.

Table 4–1 Oracle Database Background Processes

Process Name	Description
Database writer (DBW*n*)	The database writer writes modified blocks from the database buffer cache to the datafiles. Oracle Database allows a maximum of 20 database writer processes (DBW0-DBW9 and DBWa-DBWj). The initialization parameter `DB_WRITER_PROCESSES` specifies the number of DBW*n* processes. The database selects an appropriate default setting for this initialization parameter (or might adjust a user specified setting) based upon the number of CPUs and the number of processor groups. For more information about setting the `DB_WRITER_PROCESSES` initialization parameter, see the *Oracle Database Performance Tuning Guide*.
Log writer (LGWR)	The log writer process writes redo log entries to disk. Redo log entries are generated in the redo log buffer of the system global area (SGA), and LGWR writes the redo log entries sequentially into a redo log file. If the database has a multiplexed redo log, LGWR writes the redo log entries to a group of redo log files. See Chapter 6, "Managing the Redo Log" for information about the log writer process.
Checkpoint (CKPT)	At specific times, all modified database buffers in the system global area are written to the datafiles by DBW*n*. This event is called a checkpoint. The checkpoint process is responsible for signalling DBW*n* at checkpoints and updating all the datafiles and control files of the database to indicate the most recent checkpoint.
System monitor (SMON)	The system monitor performs recovery when a failed instance starts up again. In a Real Application Clusters database, the SMON process of one instance can perform instance recovery for other instances that have failed. SMON also cleans up temporary segments that are no longer in use and recovers dead transactions skipped during system failure and instance recovery because of file-read or offline errors. These transactions are eventually recovered by SMON when the tablespace or file is brought back online. SMON also coalesces free extents within the database dictionary-managed tablespaces to make free space contiguous and easier to allocate. See "Coalescing Free Space in Dictionary-Managed Tablespaces" on page 8-14).
Process monitor (PMON)	The process monitor performs process recovery when a user process fails. PMON is responsible for cleaning up the cache and freeing resources that the process was using. PMON also checks on the dispatcher processes (described later in this table) and server processes and restarts them if they have failed.

Table 4–1 (Cont.) Oracle Database Background Processes

Process Name	Description
Archiver (ARC*n*)	One or more archiver processes copy the redo log files to archival storage when they are full or a log switch occurs. Archiver processes are the subject of Chapter 7, "Managing Archived Redo Logs".
Recoverer (RECO)	The recoverer process is used to resolve distributed transactions that are pending due to a network or system failure in a distributed database. At timed intervals, the local RECO attempts to connect to remote databases and automatically complete the commit or rollback of the local portion of any pending distributed transactions. For information about this process and how to start it, see Chapter 33, "Managing Distributed Transactions".
Dispatcher (D*nnn*)	Dispatchers are optional background processes, present only when the shared server configuration is used. Shared server was discussed previously in "Configuring Oracle Database for Shared Server" on page 4-5.
Global Cache Service (LMS)	In an Oracle Real Application Clusters environment, this process manages resources and provides inter-instance resource control.

See Also: *Oracle Database Concepts* for more information about Oracle Database background processes

Managing Processes for Parallel SQL Execution

Note: The parallel execution feature described in this section is available with the Oracle Database Enterprise Edition.

This section describes how to manage parallel processing of SQL statements. In this configuration Oracle Database can divide the work of processing an SQL statement among multiple parallel processes.

The execution of many SQL statements can be parallelized. The **degree of parallelism** is the number of parallel execution servers that can be associated with a single operation. The degree of parallelism is determined by any of the following:

- A PARALLEL clause in a statement
- For objects referred to in a query, the PARALLEL clause that was used when the object was created or altered
- A parallel hint inserted into the statement

- A default determined by the database

An example of using parallel SQL execution is contained in "Parallelizing Table Creation" on page 14-10.

The following topics are contained in this section:

- About Parallel Execution Servers
- Altering Parallel Execution for a Session

See Also:

- *Oracle Database Concepts* for a description of parallel execution
- *Oracle Database Performance Tuning Guide* for information about using parallel hints

About Parallel Execution Servers

When an instance starts up, Oracle Database creates a pool of parallel execution servers which are available for any parallel operation. A process called the **parallel execution coordinator** dispatches the execution of a pool of **parallel execution servers** and coordinates the sending of results from all of these parallel execution servers back to the user.

The parallel execution servers are enabled by default, because by default the value for PARALLEL_MAX_SERVERS initialization parameter is set >0. The processes are available for use by the various Oracle Database features that are capable of exploiting parallelism. Related initialization parameters are tuned by the database for the majority of users, but you can alter them as needed to suit your environment. For ease of tuning, some parameters can be altered dynamically.

Parallelism can be used by a number of features, including transaction recovery, replication, and SQL execution. In the case of parallel SQL execution, the topic discussed in this book, parallel server processes remain associated with a statement throughout its execution phase. When the statement is completely processed, these processes become available to process other statements.

See Also: *Oracle Data Warehousing Guide* for more information about using and tuning parallel execution, including parallel SQL execution

Altering Parallel Execution for a Session

You control parallel SQL execution for a session using the ALTER SESSION statement.

Disabling Parallel SQL Execution

You disable parallel SQL execution with an ALTER SESSION DISABLE PARALLEL DML|DDL|QUERY statement. All subsequent DML (INSERT, UPDATE, DELETE), DDL (CREATE, ALTER), or query (SELECT) operations are executed serially after such a statement is issued. They will be executed serially regardless of any PARALLEL clause associated with the statement or parallel attribute associated with the table or indexes involved.

The following statement disables parallel DDL operations:

```
ALTER SESSION DISABLE PARALLEL DDL;
```

Enabling Parallel SQL Execution

You enable parallel SQL execution with an ALTER SESSION ENABLE PARALLEL DML|DDL|QUERY statement. Subsequently, when a PARALLEL clause or parallel hint is associated with a statement, those DML, DDL, or query statements will execute in parallel. By default, parallel execution is enabled for DDL and query statements.

A DML statement can be parallelized only if you specifically issue an ALTER SESSION statement to enable parallel DML:

```
ALTER SESSION ENABLE PARALLEL DML;
```

Forcing Parallel SQL Execution

You can force parallel execution of all subsequent DML, DDL, or query statements for which parallelization is possible with the ALTER SESSION FORCE PARALLEL DML|DDL|QUERY statement. Additionally you can force a specific degree of parallelism to be in effect, overriding any PARALLEL clause associated with subsequent statements. If you do not specify a degree of parallelism in this statement, the default degree of parallelism is used. However, a degree of parallelism specified in a statement through a hint will override the degree being forced.

The following statement forces parallel execution of subsequent statements and sets the overriding degree of parallelism to 5:

```
ALTER SESSION FORCE PARALLEL DDL PARALLEL 5;
```

Managing Processes for External Procedures

External procedures are procedures written in one language that are called from another program in a different language. An example is a PL/SQL program calling one or more C routines that are required to perform special-purpose processing.

These callable routines are stored in a dynamic link library (DLL), or a libunit in the case of a Java class method, and are registered with the base language. Oracle Database provides a special-purpose interface, the **call specification** (call spec), that enables users to call external procedures from other languages.

To call an external procedure, an application alerts a network listener process, which in turn starts an external procedure agent. The default name of the agent is `extproc`, and this agent must reside on the same computer as the database server. Using the network connection established by the listener, the application passes to the external procedure agent the name of the DLL or libunit, the name of the external procedure, and any relevant parameters. The external procedure agent then loads, DLL or libunit, runs the external procedure, and passes back to the application any values returned by the external procedure.

To control access to DLLs, the database administrator grants execute privileges on the appropriate DLLs to application developers. The application developers write the external procedures and grant execute privilege on specific external procedures to other users.

Note: The external library (DLL file) must be statically linked. In other words, it must not reference any external symbols from other external libraries (DLL files). Oracle Database does not resolve such symbols, so they can cause your external procedure to fail.

The environment for calling external procedures, consisting of `tnsnames.ora` and `listener.ora` entries, is configured by default during the installation of your database. You may need to perform additional network configuration steps for a higher level of security. These steps are documented in the *Oracle Net Services Administrator's Guide*.

See Also: *Oracle Database Application Developer's Guide - Fundamentals* for information about external procedures

Terminating Sessions

Sometimes it is necessary to terminate current user sessions. For example, you might want to perform an administrative operation and need to terminate all nonadministrative sessions. This section describes the various aspects of terminating sessions, and contains the following topics:

- Identifying Which Session to Terminate
- Terminating an Active Session
- Terminating an Inactive Session

When a session is terminated, any active transactions of the session are rolled back, and resources held by the session (such as locks and memory areas) are immediately released and available to other sessions.

You terminate a current session using the SQL statement `ALTER SYSTEM KILL SESSION`. The following statement terminates the session whose system identifier is 7 and serial number is 15:

```
ALTER SYSTEM KILL SESSION '7,15';
```

Identifying Which Session to Terminate

To identify which session to terminate, specify the session index number and serial number. To identify the system identifier (SID) and serial number of a session, query the `V$SESSION` dynamic performance view. For example, the following query identifies all sessions for the user `jward`:

```
SELECT SID, SERIAL#, STATUS
  FROM V$SESSION
  WHERE USERNAME = 'JWARD';

SID    SERIAL#    STATUS
-----  ---------  --------
    7         15  ACTIVE
   12         63  INACTIVE
```

A session is `ACTIVE` when it is making a SQL call to Oracle Database. A session is `INACTIVE` if it is not making a SQL call to the database.

> **See Also:** *Oracle Database Reference* for a description of the status values for a session

Terminating an Active Session

If a user session is processing a transaction (ACTIVE status) when you terminate the session, the transaction is rolled back and the user immediately receives the following message:

```
ORA-00028: your session has been killed
```

If, after receiving the ORA-00028 message, a user submits additional statements before reconnecting to the database, Oracle Database returns the following message:

```
ORA-01012: not logged on
```

An active session cannot be interrupted when it is performing network I/O or rolling back a transaction. Such a session cannot be terminated until the operation completes. In this case, the session holds all resources until it is terminated. Additionally, the session that issues the ALTER SYSTEM statement to terminate a session waits up to 60 seconds for the session to be terminated. If the operation that cannot be interrupted continues past one minute, the issuer of the ALTER SYSTEM statement receives a message indicating that the session has been marked to be terminated. A session marked to be terminated is indicated in V$SESSION with a status of KILLED and a server that is something other than PSEUDO.

Terminating an Inactive Session

If the session is not making a SQL call to Oracle Database (is INACTIVE) when it is terminated, the ORA-00028 message is not returned immediately. The message is not returned until the user subsequently attempts to use the terminated session.

When an inactive session has been terminated, the STATUS of the session in the V$SESSION view is KILLED. The row for the terminated session is removed from V$SESSION after the user attempts to use the session again and receives the ORA-00028 message.

In the following example, an inactive session is terminated. First, V$SESSION is queried to identify the SID and SERIAL# of the session, and then the session is terminated.

```
SELECT SID,SERIAL#,STATUS,SERVER
   FROM V$SESSION
   WHERE USERNAME = 'JWARD';

SID    SERIAL#   STATUS     SERVER
-----  --------  ---------  ---------
    7        15  INACTIVE   DEDICATED
```

```
   12        63  INACTIVE   DEDICATED
2 rows selected.

ALTER SYSTEM KILL SESSION '7,15';
Statement processed.

SELECT SID, SERIAL#, STATUS, SERVER
   FROM V$SESSION
   WHERE USERNAME = 'JWARD';

SID    SERIAL#   STATUS     SERVER
-----  --------  ---------  ---------
    7        15  KILLED     PSEUDO
   12        63  INACTIVE   DEDICATED
2 rows selected.
```

Monitoring the Operation of Your Database

It is important that you monitor the operation of your database on a regular basis. Doing so not only informs you about errors that have not yet come to your attention but also gives you a better understanding of the normal operation of your database. Being familiar with normal behavior in turn helps you recognize when something is wrong.

This section describes some of the options available to you for monitoring the operation of your database.

Server-Generated Alerts

A server-generated alert is a notification from the Oracle Database server of an impending problem. The notification may contain suggestions for correcting the problem. Notifications are also provided when the problem condition has been cleared.

Alerts are automatically generated when a problem occurs or when data does not match expected values for metrics, such as the following:

- Physical Reads Per Sec
- User Commits Per Sec
- SQL Service Response Time

Server-generated alerts can be based on threshold levels or can issue simply because an event has occurred. Threshold-based alerts can be triggered at both threshold

warning and critical levels. The value of these levels can be customer-defined or internal values, and some alerts have default threshold levels which you can change if appropriate. For example, by default a server-generated alert is generated for tablespace space usage when the percentage of space usage exceeds either the 85% warning or 97% critical threshold level. Examples of alerts not based on threshold levels are:

- `Snapshot Too Old`
- `Resumable Session Suspended`
- `Recovery Area Space Usage`

An alert message is sent to the predefined persistent queue `ALERT_QUE` owned by the user `SYS`. Oracle Enterprise Manager reads this queue and provides notifications about outstanding server alerts, and sometimes suggests actions for correcting the problem. The alerts are displayed on the Enterprise Manager console and can be configured to send email or pager notifications to selected administrators. If an alert cannot be written to the alert queue, a message about the alert is written to the Oracle Database alert log.

Background processes periodically flush the data to the Automatic Workload Repository to capture a history of metric values. The alert history table and `ALERT_QUE` are purged automatically by the system at regular intervals.

The most convenient way to set and view threshold values is to use Enterprise Manager. To manage threshold-based alerts through Enterprise Manager:

- On the **Database Home** page, click on the **Manage Metrics** link at the bottom of the page to display the **Thresholds** page.
- On the **Thresholds** page, you can edit the threshold values.

> **See Also:** *Oracle Enterprise Manager Concepts* for information about alerts available with Oracle Enterprise Manager

Using APIs to Administer Server-Generated Alerts

You can view and change threshold settings for the server alert metrics using the `SET_THRESHOLD` and `GET_THRESHOLD` procedures of the `DBMS_SERVER_ALERTS` PL/SQL package. The `DBMS_AQ` and `DBMS_AQADM` packages provide procedures for accessing and reading alert messages in the alert queue.

> **See Also:** *PL/SQL Packages and Types Reference* for information about the `DBMS_SERVER_ALERTS`, `DBMS_AQ`, and `DBMS_AQADM` packages

Setting Threshold Levels The following example shows how to set thresholds with the SET_THRESHOLD procedure for CPU time for each user call for an instance:

```
DBMS_SERVER_ALERT.SET_THRESHOLD(
 DBMS_SERVER_ALERT.CPU_TIME_PER_CALL, DBMS_SERVER_ALERT.OPERATOR_GE, '8000',
 DBMS_SERVER_ALERT.OPERATOR_GE, '10000', 1, 2, 'inst1',
 DBMS_SERVER_ALERT.OBJECT_TYPE_SERVICE, 'main.regress.rdbms.dev.us.oracle.com');
```

In this example, a warning alert is issued when CPU time exceeds 8000 microseconds for each user call and a critical alert is issued when CPU time exceeds 10,000 microseconds for each user call. The arguments include:

- CPU_TIME_PER_CALL specifies the metric identifier. For a list of support metrics, see *PL/SQL Packages and Types Reference*.
- The observation period is set to 1 minute. This period specifies the number of minutes that the condition must deviate from the threshold value before the alert is issued.
- The number of consecutive occurrences is set to 2. This number specifies how many times the metric value must violate the threshold values before the alert is generated.
- The name of the instance is set to inst1.
- The constant DBMS_ALERT.OBJECT_TYPE_SERVICE specifies the object type on which the threshold is set. In this example, the service name is main.regress.rdbms.dev.us.oracle.com.

Retrieving Threshold Information To retrieve threshold values, use the GET_THRESHOLD procedure. For example:

```
DECLARE
 warning_operator         BINARY_INTEGER;
 warning_value            VARCHAR2(60);
 critical_operator        BINARY_INTEGER;
 critical_value           VARCHAR2(60);
 observation_period       BINARY_INTEGER;
 consecutive_occurrences  BINARY_INTEGER;
BEGIN
 DBMS_SERVER_ALERT.GET_THRESHOLD(
 DBMS_SERVER_ALERT.CPU_TIME_PER_CALL, warning_operator, warning_value,
    critical_operator, critical_value, observation_period,
    consecutive_occurrences, 'inst1',
 DBMS_SERVER_ALERT.OBJECT_TYPE_SERVICE, 'main.regress.rdbms.dev.us.oracle.com');
 DBMS_OUTPUT.PUT_LINE('Warning operator:       ' || warning_operator);
```

```
 DBMS_OUTPUT.PUT_LINE('Warning value:          ' || warning_value);
 DBMS_OUTPUT.PUT_LINE('Critical operator:      ' || critical_operator);
 DBMS_OUTPUT.PUT_LINE('Critical value:         ' || critical_value);
 DBMS_OUTPUT.PUT_LINE('Observation_period:     ' || observation_period);
 DBMS_OUTPUT.PUT_LINE('Consecutive occurrences:' || consecutive_occurrences);
END;
/
```

You can also check specific threshold settings with the DBA_THRESHOLDS view. For example:

```
SELECT metrics_name, warning_value, critical_value, consecutive_occurrences
   FROM DBA_THRESHOLDS
   WHERE metrics_name LIKE '%CPU Time%';
```

Additional APIs to Manage Server-Generated Alerts If you use your own tool rather than Enterprise Manager to display alerts, you must subscribe to the ALERT_QUE, read the ALERT_QUE, and display an alert notification after setting the threshold levels for an alert. To create an agent and subscribe the agent to the ALERT_QUE, use the CREATE_AQ_AGENT and ADD_SUBSCRIBER procedures of the DBMS_AQADM package.

Next you must associate a database user with the subscribing agent, because only a user associated with the subscribing agent can access queued messages in the secure ALERT_QUE. You must also assign the enqueue privilege to the user. Use the ENABLE_DB_ACCESS and GRANT_QUEUE_PRIVILEGE procedures of the DBMS_AQADM package.

Optionally, you can register with the DBMS_AQ.REGISTER procedure to receive an asynchronous notification when an alert is enqueued to ALERT_QUE. The notification can be in the form of email, HTTP post, or PL/SQL procedure.

To read an alert message, you can use the DBMS_AQ.DEQUEUE procedure or OCIAQDeq call. After the message has been dequeued, use the DBMS_SERVER_ALERT.EXPAND_MESSAGE procedure to expand the text of the message.

Viewing Alert Data

The following dictionary views provide information about server alerts:

- DBA_THRESHOLDS lists the threshold settings defined for the instance.
- DBA_OUTSTANDING_ALERTS describes the outstanding alerts in the database.
- DBA_ALERT_HISTORY lists a history of alerts that have been cleared.
- V$ALERT_TYPES provides information such as group and type for each alert.

- V$METRICNAME contains the names, identifiers, and other information about the system metrics.
- V$METRIC and V$METRIC_HISTORY views contain system-level metric values in memory.

> **See Also:** *Oracle Database Reference* for information on static data dictionary views and dynamic performance views

Monitoring the Database Using Trace Files and the Alert File

Each server and background process can write to an associated **trace file**. When an internal error is detected by a process, it dumps information about the error to its trace file. Some of the information written to a trace file is intended for the database administrator, Other information is for Oracle Support Services. Trace file information is also used to tune applications and instances.

The **alert file**, or **alert log**, is a special trace file. The alert file of a database is a chronological log of messages and errors, and includes the following items:

- All internal errors (ORA-600), block corruption errors (ORA-1578), and deadlock errors (ORA-60) that occur
- Administrative operations, such as CREATE, ALTER, and DROP statements and STARTUP, SHUTDOWN, and ARCHIVELOG statements
- Messages and errors relating to the functions of shared server and dispatcher processes
- Errors occurring during the automatic refresh of a materialized view
- The values of all initialization parameters that had nondefault values at the time the database and instance start

Oracle Database uses the alert file to record these operations as an alternative to displaying the information on an operator's console (although some systems also display information on the console). If an operation is successful, a "completed" message is written in the alert file, along with a timestamp.

Initialization parameters controlling the location and size of trace files are:

- BACKGROUND_DUMP_DEST
- USER_DUMP_DEST
- MAX_DUMP_FILE_SIZE

These parameters are discussed in the sections that follow.

> **See Also:** *Oracle Database Reference* for information about initialization parameters that control the writing to trace files

Using the Trace Files

Check the alert file and other trace files of an instance periodically to learn whether the background processes have encountered errors. For example, when the log writer process (LGWR) cannot write to a member of a log group, an error message indicating the nature of the problem is written to the LGWR trace file and the database alert file. Such an error message means that a media or I/O problem has occurred and should be corrected immediately.

Oracle Database also writes values of initialization parameters to the alert file, in addition to other important statistics.

Specifying the Location of Trace Files

All trace files for background processes and the alert file are written to the directory specified by the initialization parameter `BACKGROUND_DUMP_DEST`. All trace files for server processes are written to the directory specified by the initialization parameter `USER_DUMP_DEST`. The names of trace files are operating system specific, but each file usually includes the name of the process writing the file (such as LGWR and RECO).

> **See Also:** Your operating system specific Oracle documentation for information about the names of trace files

Controlling the Size of Trace Files

You can control the maximum size of all trace files (excluding the alert file) using the initialization parameter `MAX_DUMP_FILE_SIZE`, which limits the file to the specified number of operating system blocks. To control the size of an alert file, you must manually delete the file when you no longer need it. Otherwise the database continues to append to the file.

You can safely delete the alert file while the instance is running, although you should consider making an archived copy of it first. This archived copy could prove valuable if you should have a future problem that requires investigating the history of an instance.

Controlling When Oracle Database Writes to Trace Files

Background processes always write to a trace file when appropriate. In the case of the ARC*n* background process, it is possible, through an initialization parameter, to

control the amount and type of trace information that is produced. This behavior is described in "Controlling Trace Output Generated by the Archivelog Process" on page 7-18. Other background processes do not have this flexibility.

Trace files are written on behalf of server processes whenever internal errors occur. Additionally, setting the initialization parameter SQL_TRACE = TRUE causes the SQL trace facility to generate performance statistics for the processing of all SQL statements for an instance and write them to the USER_DUMP_DEST directory.

Optionally, you can request that trace files be generated for server processes. Regardless of the current value of the SQL_TRACE initialization parameter, each session can enable or disable trace logging on behalf of the associated server process by using the SQL statement ALTER SESSION SET SQL_TRACE. This example enables the SQL trace facility for a specific session:

```
ALTER SESSION SET SQL_TRACE TRUE;
```

Caution: The SQL trace facility for server processes can cause significant system overhead resulting in severe performance impact, so you should enable this feature only when collecting statistics.

Use the DBMS_SESSION or the DBMS_MONITOR package if you want to control SQL tracing for a session.

Reading the Trace File for Shared Server Sessions

If shared server is enabled, each session using a dispatcher is routed to a shared server process, and trace information is written to the server trace file only if the session has enabled tracing (or if an error is encountered). Therefore, to track tracing for a specific session that connects using a dispatcher, you might have to explore several shared server trace files. To help you, Oracle provides a command line utility program, trcsess, which consolidates all trace information pertaining to a user session in one place and orders the information by time.

See Also: *Oracle Database Performance Tuning Guide* for information about using the SQL trace facility and using TKPROF and trcsess to interpret the generated trace files

Monitoring Locks

Locks are mechanisms that prevent destructive interaction between transactions accessing the same resource. The resources can be either user objects, such as tables and rows, or system objects not visible to users, such as shared data structures in memory and data dictionary rows. Oracle Database automatically obtains and manages necessary locks when executing SQL statements, so you need not be concerned with such details. However, the database also lets you lock data manually.

A deadlock can occur when two or more users are waiting for data locked by each other. Deadlocks prevent some transactions from continuing to work. Oracle Database automatically detects deadlock situations and resolves them by rolling back one of the statements involved in the deadlock, thereby releasing one set of the conflicting row locks.

Oracle Database is designed to avoid deadlocks, and they are not common. Most often they occur when transactions explicitly override the default locking of the database. Deadlocks can affect the performance of your database, so Oracle provides some scripts and views that enable you to monitor locks.

The `utllockt.sql` script displays, in a tree fashion, the sessions in the system that are waiting for locks and the locks that they are waiting for. The location of this script file is operating system dependent.

A second script, `catblock.sql`, creates the lock views that `utllockt.sql` needs, so you must run it before running `utllockt.sql`.

The following views can help you to monitor locks:

View	Description
V$LOCK	Lists the locks currently held by Oracle Database and outstanding requests for a lock or latch
DBA_BLOCKERS	Displays a session if it is holding a lock on an object for which another session is waiting
DBA_WAITERS	Displays a session if it is waiting for a locked object
DBA_DDL_LOCKS	Lists all DDL locks held in the database and all outstanding requests for a DDL lock
DBA_DML_LOCKS	Lists all DML locks held in the database and all outstanding requests for a DML lock
DBA_LOCK	Lists all locks or latches held in the database and all outstanding requests for a lock or latch

View	Description
DBA_LOCK_INTERNAL	Displays a row for each lock or latch that is being held, and one row for each outstanding request for a lock or latch

See Also:

- *Oracle Database Reference* contains detailed descriptions of these views
- *Oracle Database Concepts* contains more information about locks.

Monitoring Wait Events

Wait events are statistics that are incremented by a server process to indicate that it had to wait for an event to complete before being able to continue processing. A session could wait for a variety of reasons, including waiting for more input, waiting for the operating system to complete a service such as a disk write, or it could wait for a lock or latch.

When a session is waiting for resources, it is not doing any useful work. A large number of waits is a source of concern. Wait event data reveals various symptoms of problems that might be affecting performance, such as latch contention, buffer contention, and I/O contention.

Oracle provides several views that display wait event statistics. A discussion of these views and their role in instance tuning is contained in *Oracle Database Performance Tuning Guide.*

Process and Session Views

This section lists some of the data dictionary views that you can use to monitor an Oracle Database instance. These views are general in their scope. Other views, more specific to a process, are discussed in the section of this book where the process is described.

View	Description
V$PROCESS	Contains information about the currently active processes
V$LOCKED_OBJECT	Lists all locks acquired by every transaction on the system
V$SESSION	Lists session information for each current session
V$SESS_IO	Contains I/O statistics for each user session

View	Description
V$SESSION_LONGOPS	Displays the status of various operations that run for longer than 6 seconds (in absolute time). These operations currently include many backup and recovery functions, statistics gathering, and query execution. More operations are added for every Oracle Database release.
V$SESSION_WAIT	Lists the resources or events for which active sessions are waiting
V$SYSSTAT	Contains session statistics
V$RESOURCE_LIMIT	Provides information about current and maximum global resource utilization for some system resources
V$SQLAREA	Contains statistics about shared SQL area and contains one row for each SQL string. Also provides statistics about SQL statements that are in memory, parsed, and ready for execution
V$LATCH	Contains statistics for nonparent latches and summary statistics for parent latches

See Also: *Oracle Database Reference* contains detailed descriptions of these views

Part II

Oracle Database Structure and Storage

Part II describes database structure in terms of its storage components and how to create and manage those components. It contains the following chapters:

- Chapter 5, "Managing Control Files"
- Chapter 6, "Managing the Redo Log"
- Chapter 7, "Managing Archived Redo Logs"
- Chapter 8, "Managing Tablespaces"
- Chapter 9, "Managing Datafiles and Tempfiles"
- Chapter 10, "Managing the Undo Tablespace"

5

Managing Control Files

This chapter explains how to create and maintain the control files for your database and contains the following topics:

- What Is a Control File?
- Guidelines for Control Files
- Creating Control Files
- Troubleshooting After Creating Control Files
- Backing Up Control Files
- Recovering a Control File Using a Current Copy
- Dropping Control Files
- Displaying Control File Information

> **See Also:** Part III, "Automated File and Storage Management" for information about creating control files that are both created and managed by the Oracle Database server

What Is a Control File?

Every Oracle Database has a **control file**, which is a small binary file that records the physical structure of the database. The control file includes:

- The database name
- Names and locations of associated datafiles and redo log files
- The timestamp of the database creation
- The current log sequence number

- Checkpoint information

The control file must be available for writing by the Oracle Database server whenever the database is open. Without the control file, the database cannot be mounted and recovery is difficult.

The control file of an Oracle Database is created at the same time as the database. By default, at least one copy of the control file is created during database creation. On some operating systems the default is to create multiple copies. You should create two or more copies of the control file during database creation. You can also create control files later, if you lose control files or want to change particular settings in the control files.

Guidelines for Control Files

This section describes guidelines you can use to manage the control files for a database, and contains the following topics:

- Provide Filenames for the Control Files
- Multiplex Control Files on Different Disks
- Back Up Control Files
- Manage the Size of Control Files

Provide Filenames for the Control Files

You specify control file names using the `CONTROL_FILES` initialization parameter in the database initialization parameter file (see "Creating Initial Control Files" on page 5-4). The instance recognizes and opens all the listed file during startup, and the instance writes to and maintains all listed control files during database operation.

If you do not specify files for `CONTROL_FILES` before database creation:

- If you are not using Oracle-managed files, then the database creates a control file and uses a default filename. The default name is operating system specific.
- If you are using Oracle-managed files, then the initialization parameters you set to enable that feature determine the name and location of the control files, as described in Chapter 11, "Using Oracle-Managed Files".

Multiplex Control Files on Different Disks

Every Oracle Database should have at least two control files, each stored on a different physical disk. If a control file is damaged due to a disk failure, the associated instance must be shut down. Once the disk drive is repaired, the damaged control file can be restored using the intact copy of the control file from the other disk and the instance can be restarted. In this case, no media recovery is required.

The behavior of multiplexed control files is this:

- The database writes to all filenames listed for the initialization parameter `CONTROL_FILES` in the database initialization parameter file.
- The database reads only the first file listed in the `CONTROL_FILES` parameter during database operation.
- If any of the control files become unavailable during database operation, the instance becomes inoperable and should be aborted.

> **Note:** Oracle strongly recommends that your database has a minimum of two control files and that they are located on separate physical disks.

One way to multiplex control files is to store a control file copy on every disk drive that stores members of redo log groups, if the redo log is multiplexed. By storing control files in these locations, you minimize the risk that all control files and all groups of the redo log will be lost in a single disk failure.

Back Up Control Files

It is very important that you back up your control files. This is true initially, and every time you change the physical structure of your database. Such structural changes include:

- Adding, dropping, or renaming datafiles
- Adding or dropping a tablespace, or altering the read / write state of the tablespace
- Adding or dropping redo log files or groups

The methods for backing up control files are discussed in "Backing Up Control Files" on page 5-10.

Manage the Size of Control Files

The main determinants of the size of a control file are the values set for the MAXDATAFILES, MAXLOGFILES, MAXLOGMEMBERS, MAXLOGHISTORY, and MAXINSTANCES parameters in the CREATE DATABASE statement that created the associated database. Increasing the values of these parameters increases the size of a control file of the associated database.

See Also:

- Your operating system specific Oracle documentation contains more information about the maximum control file size.
- *Oracle Database SQL Reference* for a description of the CREATE DATABASE statement

Creating Control Files

This section describes ways to create control files, and contains the following topics:

- Creating Initial Control Files
- Creating Additional Copies, Renaming, and Relocating Control Files
- Creating New Control Files

Creating Initial Control Files

The initial control files of an Oracle Database are created when you issue the CREATE DATABASE statement. The names of the control files are specified by the CONTROL_FILES parameter in the initialization parameter file used during database creation. The filenames specified in CONTROL_FILES should be fully specified and are operating system specific. The following is an example of a CONTROL_FILES initialization parameter:

```
CONTROL_FILES = (/u01/oracle/prod/control01.ctl,
                 /u02/oracle/prod/control02.ctl,
                 /u03/oracle/prod/control03.ctl)
```

If files with the specified names currently exist at the time of database creation, you must specify the CONTROLFILE REUSE clause in the CREATE DATABASE statement, or else an error occurs. Also, if the size of the old control file differs from the SIZE parameter of the new one, you cannot use the REUSE clause.

The size of the control file changes between some releases of Oracle Database, as well as when the number of files specified in the control file changes. Configuration

parameters such as MAXLOGFILES, MAXLOGMEMBERS, MAXLOGHISTORY, MAXDATAFILES, and MAXINSTANCES affect control file size.

You can subsequently change the value of the CONTROL_FILES initialization parameter to add more control files or to change the names or locations of existing control files.

> **See Also:** Your operating system specific Oracle documentation contains more information about specifying control files.

Creating Additional Copies, Renaming, and Relocating Control Files

You can create an additional control file copy for multiplexing by copying an existing control file to a new location and adding the file name to the list of control files. Similarly, you rename an existing control file by copying the file to its new name or location, and changing the file name in the control file list. In both cases, to guarantee that control files do not change during the procedure, shut down the database before copying the control file.

To add a multiplexed copy of the current control file or to rename a control file:

1. Shut down the database.
2. Copy an existing control file to a new location, using operating system commands.
3. Edit the CONTROL_FILES parameter in the database initialization parameter file to add the new control file name, or to change the existing control filename.
4. Restart the database.

Creating New Control Files

This section discusses when and how to create new control files.

When to Create New Control Files

It is necessary for you to create new control files in the following situations:

- All control files for the database have been permanently damaged and you do not have a control file backup.
- You want to change one of the permanent database parameter settings originally specified in the CREATE DATABASE statement. These settings include the database name and the following parameters: MAXLOGFILES, MAXLOGMEMBERS, MAXLOGHISTORY, MAXDATAFILES, and MAXINSTANCES.

For example, you would change a database name if it conflicted with another database name in a distributed environment, or you would change the value of MAXLOGFILES if the original setting is too low.

Note: You can change the database name and DBID (internal database identifier) using the DBNEWID utility. See *Oracle Database Utilities* for information about using this utility.

The CREATE CONTROLFILE Statement

You can create a new control file for a database using the CREATE CONTROLFILE statement. The following statement creates a new control file for the prod database (a database that formerly used a different database name):

```
CREATE CONTROLFILE
   SET DATABASE prod
   LOGFILE GROUP 1 ('/u01/oracle/prod/redo01_01.log',
                    '/u01/oracle/prod/redo01_02.log'),
           GROUP 2 ('/u01/oracle/prod/redo02_01.log',
                    '/u01/oracle/prod/redo02_02.log'),
           GROUP 3 ('/u01/oracle/prod/redo03_01.log',
                    '/u01/oracle/prod/redo03_02.log')
   RESETLOGS
   DATAFILE '/u01/oracle/prod/system01.dbf' SIZE 3M,
            '/u01/oracle/prod/rbs01.dbs' SIZE 5M,
            '/u01/oracle/prod/users01.dbs' SIZE 5M,
            '/u01/oracle/prod/temp01.dbs' SIZE 5M
   MAXLOGFILES 50
   MAXLOGMEMBERS 3
   MAXLOGHISTORY 400
   MAXDATAFILES 200
   MAXINSTANCES 6
   ARCHIVELOG;
```

Cautions:

- The `CREATE CONTROLFILE` statement can potentially damage specified datafiles and redo log files. Omitting a filename can cause loss of the data in that file, or loss of access to the entire database. Use caution when issuing this statement and be sure to follow the instructions in "Steps for Creating New Control Files".
- If the database had forced logging enabled before creating the new control file, and you want it to continue to be enabled, then you must specify the `FORCE LOGGING` clause in the `CREATE CONTROLFILE` statement. See "Specifying FORCE LOGGING Mode" on page 2-26.

See Also: *Oracle Database SQL Reference* describes the complete syntax of the `CREATE CONTROLFILE` statement

Steps for Creating New Control Files

Complete the following steps to create a new control file.

1. Make a list of all datafiles and redo log files of the database.

   If you follow recommendations for control file backups as discussed in "Backing Up Control Files" on page 5-10, you will already have a list of datafiles and redo log files that reflect the current structure of the database. However, if you have no such list, executing the following statements will produce one.

   ```
   SELECT MEMBER FROM V$LOGFILE;
   SELECT NAME FROM V$DATAFILE;
   SELECT VALUE FROM V$PARAMETER WHERE NAME = 'control_files';
   ```

   If you have no such lists and your control file has been damaged so that the database cannot be opened, try to locate all of the datafiles and redo log files that constitute the database. Any files not specified in step 5 are not recoverable once a new control file has been created. Moreover, if you omit any of the files that make up the `SYSTEM` tablespace, you might not be able to recover the database.

2. Shut down the database.

   If the database is open, shut down the database normally if possible. Use the `IMMEDIATE` or `ABORT` clauses only as a last resort.

3. Back up all datafiles and redo log files of the database.
4. Start up a new instance, but do not mount or open the database:

   ```
   STARTUP NOMOUNT
   ```

5. Create a new control file for the database using the `CREATE CONTROLFILE` statement.

   When creating a new control file, specify the `RESETLOGS` clause if you have lost any redo log groups in addition to control files. In this case, you will need to recover from the loss of the redo logs (step 8). You must specify the `RESETLOGS` clause if you have renamed the database. Otherwise, select the `NORESETLOGS` clause.
6. Store a backup of the new control file on an offline storage device. See "Backing Up Control Files" on page 5-10 for instructions for creating a backup.
7. Edit the `CONTROL_FILES` initialization parameter for the database to indicate all of the control files now part of your database as created in step 5 (not including the backup control file). If you are renaming the database, edit the `DB_NAME` parameter in your instance parameter file to specify the new name.
8. Recover the database if necessary. If you are not recovering the database, skip to step 9.

   If you are creating the control file as part of recovery, recover the database. If the new control file was created using the `NORESETLOGS` clause (step 5), you can recover the database with complete, closed database recovery.

   If the new control file was created using the `RESETLOGS` clause, you must specify `USING BACKUP CONTROL FILE`. If you have lost online or archived redo logs or datafiles, use the procedures for recovering those files.

   > **See Also:** *Oracle Database Backup and Recovery Basics* and *Oracle Database Backup and Recovery Advanced User's Guide* for information about recovering your database and methods of recovering a lost control file

9. Open the database using one of the following methods:

   - If you did not perform recovery, or you performed complete, closed database recovery in step 8, open the database normally.

     ```
     ALTER DATABASE OPEN;
     ```

- If you specified RESETLOGS when creating the control file, use the ALTER DATABASE statement, indicating RESETLOGS.

  ```
  ALTER DATABASE OPEN RESETLOGS;
  ```

The database is now open and available for use.

Troubleshooting After Creating Control Files

After issuing the CREATE CONTROLFILE statement, you may encounter some errors. This section describes the most common control file errors:

- Checking for Missing or Extra Files
- Handling Errors During CREATE CONTROLFILE

Checking for Missing or Extra Files

After creating a new control file and using it to open the database, check the alert file to see if the database has detected inconsistencies between the data dictionary and the control file, such as a datafile in the data dictionary includes that the control file does not list.

If a datafile exists in the data dictionary but not in the new control file, the database creates a placeholder entry in the control file under the name MISSING*nnnn*, where *nnnn* is the file number in decimal. MISSING*nnnn* is flagged in the control file as being offline and requiring media recovery.

If the actual datafile corresponding to MISSING*nnnn* is read-only or offline normal, then you can make the datafile accessible by renaming MISSING*nnnn* to the name of the actual datafile. If MISSING*nnnn* corresponds to a datafile that was not read-only or offline normal, then you cannot use the rename operation to make the datafile accessible, because the datafile requires media recovery that is precluded by the results of RESETLOGS. In this case, you must drop the tablespace containing the datafile.

Conversely, if a datafile listed in the control file is not present in the data dictionary, then the database removes references to it from the new control file. In both cases, the database includes an explanatory message in the alert file to let you know what was found.

Handling Errors During CREATE CONTROLFILE

If Oracle Database sends you an error (usually error ORA-01173, ORA-01176, ORA-01177, ORA-01215, or ORA-01216) when you attempt to mount and open the database after creating a new control file, the most likely cause is that you omitted a file from the CREATE CONTROLFILE statement or included one that should not have been listed. In this case, you should restore the files you backed up in step 3 on page 5-8 and repeat the procedure from step 4, using the correct filenames.

Backing Up Control Files

Use the ALTER DATABASE BACKUP CONTROLFILE statement to back up your control files. You have two options:

1. Back up the control file to a binary file (duplicate of existing control file) using the following statement:

   ```
   ALTER DATABASE BACKUP CONTROLFILE TO '/oracle/backup/control.bkp';
   ```

2. Produce SQL statements that can later be used to re-create your control file:

   ```
   ALTER DATABASE BACKUP CONTROLFILE TO TRACE;
   ```

   This command writes a SQL script to the database trace file where it can be captured and edited to reproduce the control file.

   See Also: *Oracle Database Backup and Recovery Basics* for more information on backing up your control files

Recovering a Control File Using a Current Copy

This section presents ways that you can recover your control file from a current backup or from a multiplexed copy.

Recovering from Control File Corruption Using a Control File Copy

This procedure assumes that one of the control files specified in the CONTROL_FILES parameter is corrupted, that the control file directory is still accessible, and that you have a multiplexed copy of the control file.

1. With the instance shut down, use an operating system command to overwrite the bad control file with a good copy:

```
% cp /u03/oracle/prod/control03.ctl  /u02/oracle/prod/control02.ctl
```

2. Start SQL*Plus and open the database:

```
SQL> STARTUP
```

Recovering from Permanent Media Failure Using a Control File Copy

This procedure assumes that one of the control files specified in the CONTROL_FILES parameter is inaccessible due to a permanent media failure and that you have a multiplexed copy of the control file.

1. With the instance shut down, use an operating system command to copy the current copy of the control file to a new, accessible location:

```
% cp /u01/oracle/prod/control01.ctl  /u04/oracle/prod/control03.ctl
```

2. Edit the CONTROL_FILES parameter in the initialization parameter file to replace the bad location with the new location:

```
CONTROL_FILES = (/u01/oracle/prod/control01.ctl,
                 /u02/oracle/prod/control02.ctl,
                 /u04/oracle/prod/control03.ctl)
```

3. Start SQL*Plus and open the database:

```
SQL> STARTUP
```

If you have multiplexed control files, you can get the database started up quickly by editing the CONTROL_FILES initialization parameter. Remove the bad control file from CONTROL_FILES setting and you can restart the database immediately. Then you can perform the reconstruction of the bad control file and at some later time shut down and restart the database after editing the CONTROL_FILES initialization parameter to include the recovered control file.

Dropping Control Files

You want to drop control files from the database, for example, if the location of a control file is no longer appropriate. Remember that the database should have at least two control files at all times.

1. Shut down the database.
2. Edit the CONTROL_FILES parameter in the database initialization parameter file to delete the old control file name.

3. Restart the database.

> **Note:** This operation does not physically delete the unwanted control file from the disk. Use operating system commands to delete the unnecessary file after you have dropped the control file from the database.

Displaying Control File Information

The following views display information about control files:

View	Description
V$DATABASE	Displays database information from the control file
V$CONTROLFILE	Lists the names of control files
V$CONTROLFILE_RECORD_SECTION	Displays information about control file record sections
V$PARAMETER	Displays the names of control files as specified in the CONTROL_FILES initialization parameter

This example lists the names of the control files.

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

NAME
------------------------------------
/u01/oracle/prod/control01.ctl
/u02/oracle/prod/control02.ctl
/u03/oracle/prod/control03.ctl
```

6

Managing the Redo Log

This chapter explains how to manage the online redo log. The current redo log is always online, unlike archived copies of a redo log. Therefore, the online redo log is usually referred to as simply the **redo log**.

This chapter contains the following topics:

- What Is the Redo Log?
- Planning the Redo Log
- Creating Redo Log Groups and Members
- Relocating and Renaming Redo Log Members
- Dropping Redo Log Groups and Members
- Forcing Log Switches
- Verifying Blocks in Redo Log Files
- Clearing a Redo Log File
- Viewing Redo Log Information

> **See Also:** Part III, "Automated File and Storage Management" for information about redo log files that are both created and managed by the Oracle Database server

What Is the Redo Log?

The most crucial structure for recovery operations is the **redo log**, which consists of two or more preallocated files that store all changes made to the database as they occur. Every instance of an Oracle Database has an associated redo log to protect the database in case of an instance failure.

Redo Threads

Each database instance has its own **redo log groups**. These redo log groups, multiplexed or not, are called an instance **thread** of redo. In typical configurations, only one database instance accesses an Oracle Database, so only one thread is present. In an Oracle Real Application Clusters environment, however, two or more instances concurrently access a single database and each instance has its own thread of redo.

This chapter describes how to configure and manage the redo log on a standard single-instance Oracle Database. The thread number can be assumed to be 1 in all discussions and examples of statements. For information about redo log groups in a Real Application Environment, please refer to *Oracle Real Application Clusters Administrator's Guide*.

Redo Log Contents

Redo log files are filled with **redo records**. A redo record, also called a **redo entry**, is made up of a group of **change vectors**, each of which is a description of a change made to a single block in the database. For example, if you change a salary value in an employee table, you generate a redo record containing change vectors that describe changes to the data segment block for the table, the undo segment data block, and the transaction table of the undo segments.

Redo entries record data that you can use to reconstruct all changes made to the database, including the undo segments. Therefore, the redo log also protects rollback data. When you recover the database using redo data, the database reads the change vectors in the redo records and applies the changes to the relevant blocks.

Redo records are buffered in a circular fashion in the redo log buffer of the SGA (see "How Oracle Database Writes to the Redo Log" on page 6-3) and are written to one of the redo log files by the Log Writer (LGWR) database background process. Whenever a transaction is committed, LGWR writes the transaction redo records from the redo log buffer of the SGA to a redo log file, and assigns a **system change number** (SCN) to identify the redo records for each committed transaction. Only when all redo records associated with a given transaction are safely on disk in the online logs is the user process notified that the transaction has been committed.

Redo records can also be written to a redo log file before the corresponding transaction is committed. If the redo log buffer fills, or another transaction commits, LGWR flushes all of the redo log entries in the redo log buffer to a redo log file, even though some redo records may not be committed. If necessary, the database can roll back these changes.

How Oracle Database Writes to the Redo Log

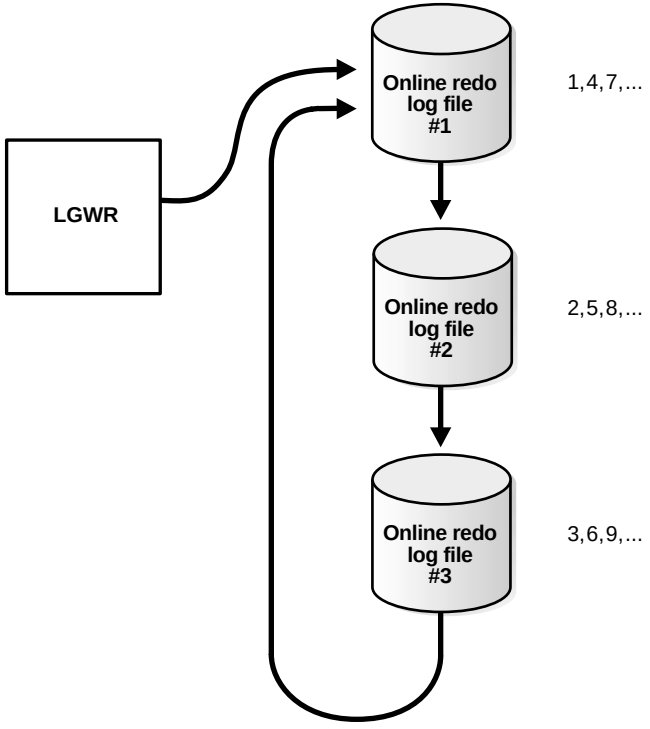
The redo log of a database consists of two or more redo log files. The database requires a minimum of two files to guarantee that one is always available for writing while the other is being archived (if the database is in ARCHIVELOG mode).

LGWR writes to redo log files in a circular fashion. When the current redo log file fills, LGWR begins writing to the next available redo log file. When the last available redo log file is filled, LGWR returns to the first redo log file and writes to it, starting the cycle again. Figure 6–1 illustrates the circular writing of the redo log file. The numbers next to each line indicate the sequence in which LGWR writes to each redo log file.

Filled redo log files are available to LGWR for reuse depending on whether archiving is enabled.

- If archiving is disabled (the database is in NOARCHIVELOG mode), a filled redo log file is available after the changes recorded in it have been written to the datafiles.
- If archiving is enabled (the database is in ARCHIVELOG mode), a filled redo log file is available to LGWR after the changes recorded in it have been written to the datafiles *and* the file has been archived.

Figure 6–1 Circular Use of Redo Log Files by LGWR



Active (Current) and Inactive Redo Log Files

Oracle Database uses only one redo log files at a time to store redo records written from the redo log buffer. The redo log file that LGWR is actively writing to is called the **current** redo log file.

Redo log files that are required for instance recovery are called **active** redo log files. Redo log files that are no longer required for instance recovery are called **inactive** redo log files.

If you have enabled archiving (the database is in `ARCHIVELOG` mode), then the database cannot reuse or overwrite an active online log file until one of the archiver background processes (ARC*n*) has archived its contents. If archiving is disabled (the database is in `NOARCHIVELOG` mode), then when the last redo log file is full, LGWR continues by overwriting the first available active file.

Log Switches and Log Sequence Numbers

A **log switch** is the point at which the database stops writing to one redo log file and begins writing to another. Normally, a log switch occurs when the current redo log file is completely filled and writing must continue to the next redo log file. However, you can configure log switches to occur at regular intervals, regardless of whether the current redo log file is completely filled. You can also force log switches manually.

Oracle Database assigns each redo log file a new **log sequence number** every time a log switch occurs and LGWR begins writing to it. When the database archives redo log files, the archived log retains its log sequence number. A redo log file that is cycled back for use is given the next available log sequence number.

Each online or archived redo log file is uniquely identified by its log sequence number. During crash, instance, or media recovery, the database properly applies redo log files in ascending order by using the log sequence number of the necessary archived and redo log files.

Planning the Redo Log

This section provides guidelines you should consider when configuring a database instance redo log and contains the following topics:

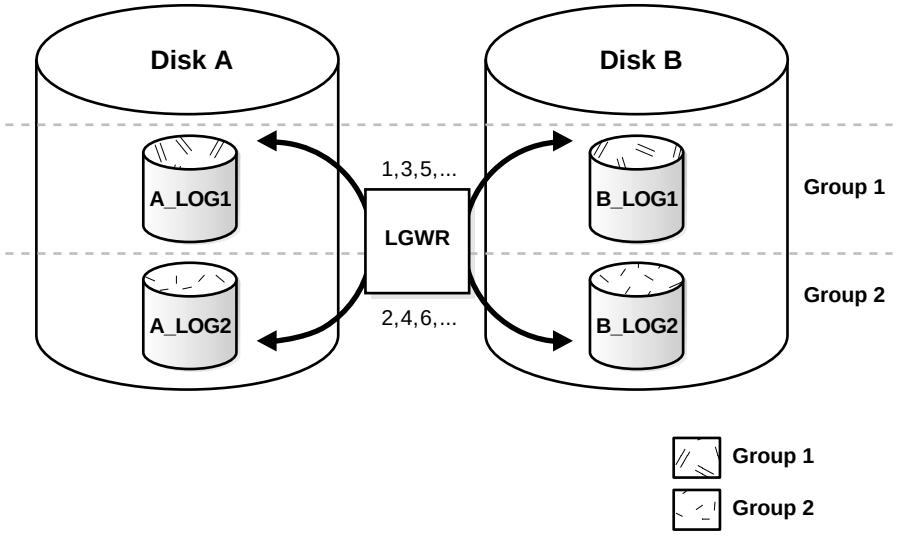
- Multiplexing Redo Log Files
- Placing Redo Log Members on Different Disks
- Setting the Size of Redo Log Members
- Choosing the Number of Redo Log Files
- Controlling Archive Lag

Multiplexing Redo Log Files

Oracle Database lets you **multiplex** the redo log files of an instance to safeguard against damage to any single file. When redo log files are multiplexed, LGWR concurrently writes the same redo log information to multiple identical redo log files, thereby eliminating a single point of redo log failure.

Note: Oracle recommends that you multiplex your redo log files. The loss of the log file data can be catastrophic if recovery is required.

Figure 6–2 Multiplexed Redo Log Files



Multiplexed redo log files are called **groups**. Each redo log file in a group is called a **member**. In Figure 6–2, A_LOG1 and B_LOG1 are both members of Group 1, A_LOG2 and B_LOG2 are both members of Group 2, and so forth. Each member in a group must be exactly the same size.

Each member of a log file group is concurrently active--that is, concurrently written to by LGWR--as indicated by the identical log sequence numbers assigned by LGWR. In Figure 6–2, first LGWR writes concurrently to both A_LOG1 and B_LOG1. Then it writes concurrently to both A_LOG2 and B_LOG2, and so on. LGWR never writes concurrently to members of different groups (for example, to A_LOG1 and B_LOG2).

Responding to Redo Log Failure

Whenever LGWR cannot write to a member of a group, the database marks that member as INVALID and writes an error message to the LGWR trace file and to the database alert file to indicate the problem with the inaccessible files. The specific reaction of LGWR when a redo log member is unavailable depends on the reason for the lack of availability, as summarized in the table that follows.

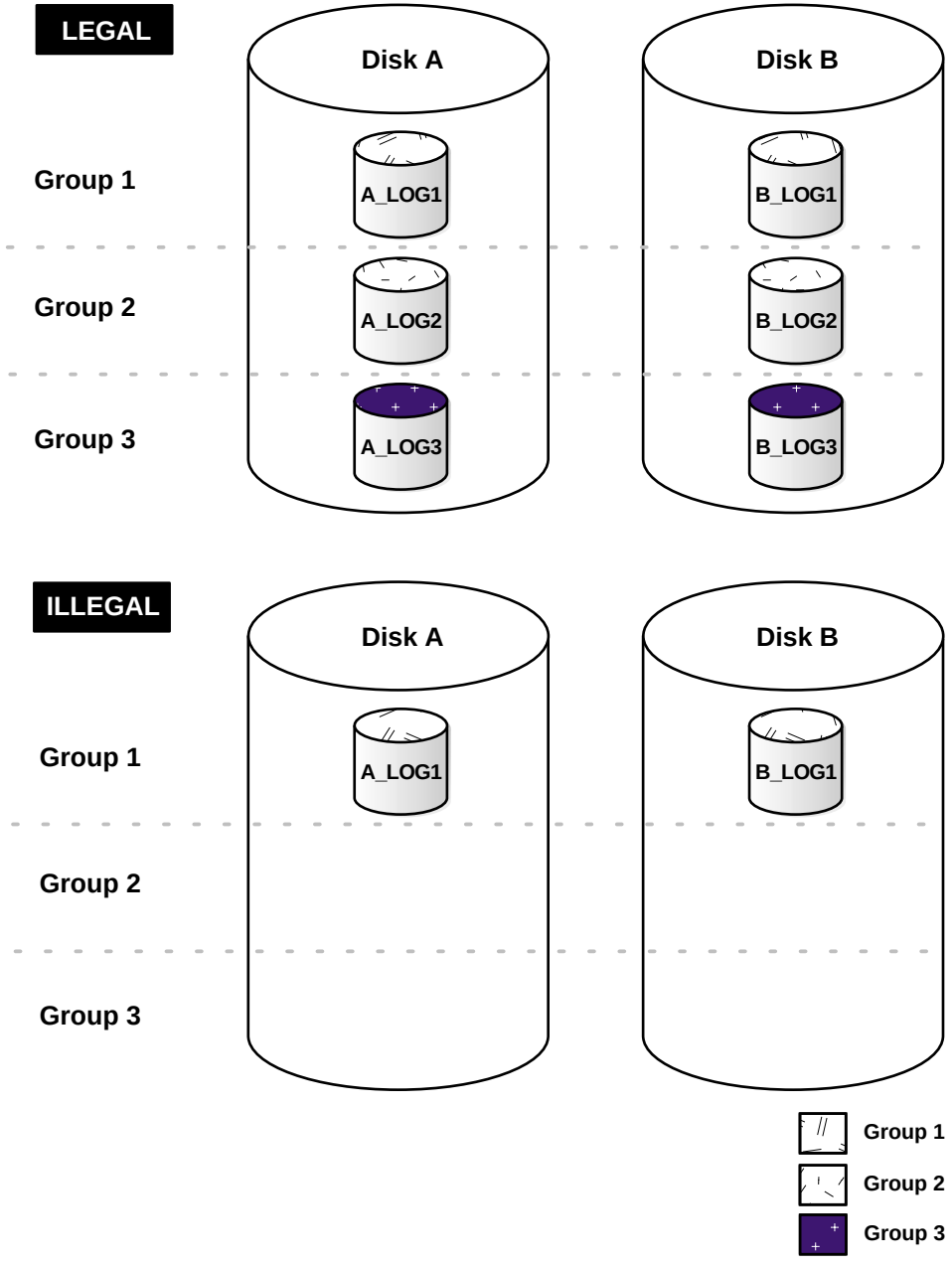
Condition	LGWR Action
LGWR can successfully write to at least one member in a group	Writing proceeds as normal. LGWR writes to the available members of a group and ignores the unavailable members.
LGWR cannot access the next group at a log switch because the group needs to be archived	Database operation temporarily halts until the group becomes available or until the group is archived.
All members of the next group are inaccessible to LGWR at a log switch because of media failure	Oracle Database returns an error, and the database instance shuts down. In this case, you may need to perform media recovery on the database from the loss of a redo log file. If the database checkpoint has moved beyond the lost redo log, media recovery is not necessary, because the database has saved the data recorded in the redo log to the datafiles. You need only drop the inaccessible redo log group. If the database did not archive the bad log, use `ALTER DATABASE CLEAR UNARCHIVED LOG` to disable archiving before the log can be dropped.
All members of a group suddenly become inaccessible to LGWR while it is writing to them	Oracle Database returns an error and the database instance immediately shuts down. In this case, you may need to perform media recovery. If the media containing the log is not actually lost--for example, if the drive for the log was inadvertently turned off--media recovery may not be needed. In this case, you need only turn the drive back on and let the database perform automatic instance recovery.

Legal and Illegal Configurations

In most cases, a multiplexed redo log should be symmetrical: all groups of the redo log should have the same number of members. However, the database does not require that a multiplexed redo log be symmetrical. For example, one group can have only one member, and other groups can have two members. This configuration protects against disk failures that temporarily affect some redo log members but leave others intact.

The only requirement for an instance redo log is that it have at least two groups. Figure 6–3 shows legal and illegal multiplexed redo log configurations. The second configuration is illegal because it has only one group.

Figure 6–3 Legal and Illegal Multiplexed Redo Log Configuration



Placing Redo Log Members on Different Disks

When setting up a multiplexed redo log, place members of a group on different physical disks. If a single disk fails, then only one member of a group becomes unavailable to LGWR and other members remain accessible to LGWR, so the instance can continue to function.

If you archive the redo log, spread redo log members across disks to eliminate contention between the LGWR and ARC*n* background processes. For example, if you have two groups of duplexed redo log members, place each member on a different disk and set your archiving destination to a fifth disk. Doing so will avoid contention between LGWR (writing to the members) and ARC*n* (reading the members).

Datafiles should also be placed on different disks from redo log files to reduce contention in writing data blocks and redo records.

Setting the Size of Redo Log Members

When setting the size of redo log files, consider whether you will be archiving the redo log. Redo log files should be sized so that a filled group can be archived to a single unit of offline storage media (such as a tape or disk), with the least amount of space on the medium left unused. For example, suppose only one filled redo log group can fit on a tape and 49% of the tape storage capacity remains unused. In this case, it is better to decrease the size of the redo log files slightly, so that two log groups could be archived on each tape.

All members of the same multiplexed redo log group must be the same size. Members of different groups can have different sizes. However, there is no advantage in varying file size between groups. If checkpoints are not set to occur between log switches, make all groups the same size to guarantee that checkpoints occur at regular intervals.

> **See Also:** Your operating system specific Oracle documentation. The default size of redo log files is operating system dependent.

Choosing the Number of Redo Log Files

The best way to determine the appropriate number of redo log files for a database instance is to test different configurations. The optimum configuration has the fewest groups possible without hampering LGWR from writing redo log information.

In some cases, a database instance may require only two groups. In other situations, a database instance may require additional groups to guarantee that a recycled group is always available to LGWR. During testing, the easiest way to determine whether the current redo log configuration is satisfactory is to examine the contents of the LGWR trace file and the database alert log. If messages indicate that LGWR frequently has to wait for a group because a checkpoint has not completed or a group has not been archived, add groups.

Consider the parameters that can limit the number of redo log files before setting up or altering the configuration of an instance redo log. The following parameters limit the number of redo log files that you can add to a database:

- The MAXLOGFILES parameter used in the CREATE DATABASE statement determines the maximum number of groups of redo log files for each database. Group values can range from 1 to MAXLOGFILES. The only way to override this upper limit is to re-create the database or its control file. Therefore, it is important to consider this limit before creating a database. If MAXLOGFILES is not specified for the CREATE DATABASE statement, then the database uses an operating system specific default value.
- The MAXLOGMEMBERS parameter used in the CREATE DATABASE statement determines the maximum number of members for each group. As with MAXLOGFILES, the only way to override this upper limit is to re-create the database or control file. Therefore, it is important to consider this limit before creating a database. If no MAXLOGMEMBERS parameter is specified for the CREATE DATABASE statement, then the database uses an operating system default value.

See Also:

- *Oracle Database Backup and Recovery Advanced User's Guide* to learn how checkpoints and the redo log impact instance recovery
- Your operating system specific Oracle documentation for the default and legal values of the MAXLOGFILES and MAXLOGMEMBERS parameters

Controlling Archive Lag

You can force all enabled redo log threads to switch their current logs at regular time intervals. In a primary / standby database configuration, changes are made available to the standby database by archiving redo logs at the primary site and then shipping them to the standby database. The changes that are being applied by

the standby database can lag behind the changes that are occurring on the primary database, because the standby database must wait for the changes in the primary database redo log to be archived (into the archived redo log) and then shipped to it. To limit this lag, you can set the ARCHIVE_LAG_TARGET initialization parameter. Setting this parameter lets you specify in seconds how long that lag can be.

Setting the ARCHIVE_LAG_TARGET Initialization Parameter

When you set the ARCHIVE_LAG_TARGET initialization parameter, you cause the database to examine an the current redo log of the instance periodically. If the following conditions are met, then the instance will switch the log:

- The current log was created prior to *n* seconds ago, and the estimated archival time for the current log is *m* seconds (proportional to the number of redo blocks used in the current log), where *n* + *m* exceeds the value of the ARCHIVE_LAG_TARGET initialization parameter.
- The current log contains redo records.

In an Oracle Real Application Clusters environment, the instance also causes other threads to switch and archive their logs if they are falling behind. This can be particularly useful when one instance in the cluster is more idle than the other instances (as when you are running a 2-node primary / secondary configuration of Oracle Real Application Clusters).

The ARCHIVE_LAG_TARGET initialization parameter specifies the target of how many seconds of redo the standby could lose in the event of a primary shutdown or failure if the Oracle Data Guard environment is not configured in a no-data-loss mode. It also provides an upper limit of how long (in seconds) the current log of the primary database can span. Because the estimated archival time is also considered, this is not the exact log switch time.

The following initialization parameter setting sets the log switch interval to 30 minutes (a typical value).

```
ARCHIVE_LAG_TARGET = 1800
```

A value of 0 disables this time-based log switching functionality. This is the default setting.

You can set the ARCHIVE_LAG_TARGET initialization parameter even if there is no standby database. For example, the ARCHIVE_LAG_TARGET parameter can be set specifically to force logs to be switched and archived.

ARCHIVE_LAG_TARGET is a dynamic parameter and can be set with the ALTER SYSTEM SET statement.

Caution: The ARCHIVE_LAG_TARGET parameter must be set to the same value in all instances of an Oracle Real Application Clusters environment. Failing to do so results in unpredictable behavior.

Factors Affecting the Setting of ARCHIVE_LAG_TARGET

Consider the following factors when determining if you want to set the ARCHIVE_LAG_TARGET parameter and in determining the value for this parameter.

- Overhead of switching (as well as archiving) logs
- How frequently normal log switches occur as a result of log full conditions
- How much redo loss is tolerated in the standby database

Setting ARCHIVE_LAG_TARGET may not be very useful if natural log switches already occur more frequently than the interval specified. However, in the case of irregularities of redo generation speed, the interval does provide an upper limit for the time range each current log covers.

If the ARCHIVE_LAG_TARGET initialization parameter is set to a very low value, there can be a negative impact on performance. This can force frequent log switches. Set the parameter to a reasonable value so as not to degrade the performance of the primary database.

Creating Redo Log Groups and Members

Plan the redo log of a database and create all required groups and members of redo log files during database creation. However, there are situations where you might want to create additional groups or members. For example, adding groups to a redo log can correct redo log group availability problems.

To create new redo log groups and members, you must have the ALTER DATABASE system privilege. A database can have up to MAXLOGFILES groups.

See Also: *Oracle Database SQL Reference* for a complete description of the ALTER DATABASE statement

Creating Redo Log Groups

To create a new group of redo log files, use the SQL statement ALTER DATABASE with the ADD LOGFILE clause.

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

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

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

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

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

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

Creating Redo Log Members

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

To create new redo log members for an existing group, use the SQL statement ALTER DATABASE with the ADD LOGFILE MEMBER clause. The following statement adds a new redo log member to redo log group number 2:

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

Notice that filenames must be specified, but sizes need not be. The size of the new members is determined from the size of the existing members of the group.

When using the ALTER DATABASE statement, you can alternatively identify the target group by specifying all of the other members of the group in the TO clause, as shown in the following example:

```
ALTER DATABASE ADD LOGFILE MEMBER '/oracle/dbs/log2c.rdo'
    TO ('/oracle/dbs/log2a.rdo', '/oracle/dbs/log2b.rdo');
```

Note: Fully specify the filenames of new log members to indicate where the operating system files should be created. Otherwise, the files will be created in either the default or current directory of the database server, depending upon your operating system. You may also note that the status of the new log member is shown as `INVALID`. This is normal and it will change to active (blank) when it is first used.

Relocating and Renaming Redo Log Members

You can use operating system commands to relocate redo logs, then use the ALTER DATABASE statement to make their new names (locations) known to the database. This procedure is necessary, for example, if the disk currently used for some redo log files is going to be removed, or if datafiles and a number of redo log files are stored on the same disk and should be separated to reduce contention.

To rename redo log members, you must have the ALTER DATABASE system privilege. Additionally, you might also need operating system privileges to copy files to the desired location and privileges to open and back up the database.

Before relocating your redo logs, or making any other structural changes to the database, completely back up the database in case you experience problems while performing the operation. As a precaution, after renaming or relocating a set of redo log files, immediately back up the database control file.

Use the following steps for relocating redo logs. The example used to illustrate these steps assumes:

- The log files are located on two disks: `diska` and `diskb`.
- The redo log is duplexed: one group consists of the members `/diska/logs/log1a.rdo` and `/diskb/logs/log1b.rdo`, and the second group consists of the members `/diska/logs/log2a.rdo` and `/diskb/logs/log2b.rdo`.
- The redo log files located on `diska` must be relocated to `diskc`. The new filenames will reflect the new location: `/diskc/logs/log1c.rdo` and `/diskc/logs/log2c.rdo`.

Steps for Renaming Redo Log Members

1. Shut down the database.

   ```
   SHUTDOWN
   ```

2. Copy the redo log files to the new location.

   Operating system files, such as redo log members, must be copied using the appropriate operating system commands. See your operating system specific documentation for more information about copying files.

   > **Note:** You can execute an operating system command to copy a file (or perform other operating system commands) without exiting SQL*Plus by using the HOST command. Some operating systems allow you to use a character in place of the word HOST. For example, you can use an exclamation point (!) in UNIX.

   The following example uses operating system commands (UNIX) to move the redo log members to a new location:

   ```
   mv /diska/logs/log1a.rdo /diskc/logs/log1c.rdo
   mv /diska/logs/log2a.rdo /diskc/logs/log2c.rdo
   ```

3. Startup the database, mount, but do not open it.

   ```
   CONNECT / as SYSDBA
   STARTUP MOUNT
   ```

4. Rename the redo log members.

   Use the ALTER DATABASE statement with the RENAME FILE clause to rename the database redo log files.

   ```
   ALTER DATABASE
     RENAME FILE '/diska/logs/log1a.rdo', '/diska/logs/log2a.rdo'
              TO '/diskc/logs/log1c.rdo', '/diskc/logs/log2c.rdo';
   ```

5. Open the database for normal operation.

   The redo log alterations take effect when the database is opened.

   ```
   ALTER DATABASE OPEN;
   ```

Dropping Redo Log Groups and Members

In some cases, you may want to drop an entire group of redo log members. For example, you want to reduce the number of groups in an instance redo log. In a different case, you may want to drop one or more specific redo log members. For

example, if a disk failure occurs, you may need to drop all the redo log files on the failed disk so that the database does not try to write to the inaccessible files. In other situations, particular redo log files become unnecessary. For example, a file might be stored in an inappropriate location.

Dropping Log Groups

To drop a redo log group, you must have the ALTER DATABASE system privilege. Before dropping a redo log group, consider the following restrictions and precautions:

- An instance requires at least two groups of redo log files, regardless of the number of members in the groups. (A group comprises one or more members.)
- You can drop a redo log group only if it is inactive. If you need to drop the current group, first force a log switch to occur.
- Make sure a redo log group is archived (if archiving is enabled) before dropping it. To see whether this has happened, use the V$LOG view.

```
SELECT GROUP#, ARCHIVED, STATUS FROM V$LOG;

   GROUP# ARC STATUS
--------- --- ----------------
        1 YES ACTIVE
        2 NO  CURRENT
        3 YES INACTIVE
        4 YES INACTIVE
```

Drop a redo log group with the SQL statement ALTER DATABASE with the DROP LOGFILE clause.

The following statement drops redo log group number 3:

```
ALTER DATABASE DROP LOGFILE GROUP 3;
```

When a redo log group is dropped from the database, and you are not using the Oracle-managed files feature, the operating system files are not deleted from disk. Rather, the control files of the associated database are updated to drop the members of the group from the database structure. After dropping a redo log group, make sure that the drop completed successfully, and then use the appropriate operating system command to delete the dropped redo log files.

When using Oracle-managed files, the cleanup of operating systems files is done automatically for you.

Dropping Redo Log Members

To drop a redo log member, you must have the ALTER DATABASE system privilege. Consider the following restrictions and precautions before dropping individual redo log members:

- It is permissible to drop redo log files so that a multiplexed redo log becomes temporarily asymmetric. For example, if you use duplexed groups of redo log files, you can drop one member of one group, even though all other groups have two members each. However, you should rectify this situation immediately so that all groups have at least two members, and thereby eliminate the single point of failure possible for the redo log.
- An instance always requires at least two valid groups of redo log files, regardless of the number of members in the groups. (A group comprises one or more members.) If the member you want to drop is the last valid member of the group, you cannot drop the member until the other members become valid. To see a redo log file status, use the V$LOGFILE view. A redo log file becomes INVALID if the database cannot access it. It becomes STALE if the database suspects that it is not complete or correct. A stale log file becomes valid again the next time its group is made the active group.
- You can drop a redo log member only if it is *not* part of an active or current group. If you want to drop a member of an active group, first force a log switch to occur.
- Make sure the group to which a redo log member belongs is archived (if archiving is enabled) before dropping the member. To see whether this has happened, use the V$LOG view.

To drop specific inactive redo log members, use the ALTER DATABASE statement with the DROP LOGFILE MEMBER clause.

The following statement drops the redo log /oracle/dbs/log3c.rdo:

```
ALTER DATABASE DROP LOGFILE MEMBER '/oracle/dbs/log3c.rdo';
```

When a redo log member is dropped from the database, the operating system file is not deleted from disk. Rather, the control files of the associated database are updated to drop the member from the database structure. After dropping a redo log file, make sure that the drop completed successfully, and then use the appropriate operating system command to delete the dropped redo log file.

To drop a member of an active group, you must first force a log switch.

Forcing Log Switches

A log switch occurs when LGWR stops writing to one redo log group and starts writing to another. By default, a log switch occurs automatically when the current redo log file group fills.

You can force a log switch to make the currently active group inactive and available for redo log maintenance operations. For example, you want to drop the currently active group, but are not able to do so until the group is inactive. You may also wish to force a log switch if the currently active group needs to be archived at a specific time before the members of the group are completely filled. This option is useful in configurations with large redo log files that take a long time to fill.

To force a log switch, you must have the `ALTER SYSTEM` privilege. Use the `ALTER SYSTEM` statement with the `SWITCH LOGFILE` clause.

The following statement forces a log switch:

```
ALTER SYSTEM SWITCH LOGFILE;
```

Verifying Blocks in Redo Log Files

You can configure the database to use checksums to verify blocks in the redo log files. If you set the initialization parameter `DB_BLOCK_CHECKSUM` to `TRUE`, the database computes a checksum for each database block when it is written to disk, including each redo log block as it is being written to the current log. The checksum is stored the header of the block.

Oracle Database uses the checksum to detect corruption in a redo log block. The database verifies the redo log block when the block is read from an archived log during recovery and when it writes the block to an archive log file. An error is raised and written to the alert log if corruption is detected.

If corruption is detected in a redo log block while trying to archive it, the system attempts to read the block from another member in the group. If the block is corrupted in all members of the redo log group, then archiving cannot proceed.

The default value of `DB_BLOCK_CHECKSUM` is `TRUE`. The value of this parameter can be changed dynamically using the `ALTER SYSTEM` statement.

Note: There is a slight overhead and decrease in database performance with `DB_BLOCK_CHECKSUM` enabled. Monitor your database performance to decide if the benefit of using data block checksums to detect corruption outweighs the performance impact.

See Also: *Oracle Database Reference* for a description of the DB_BLOCK_CHECKSUM initialization parameter

Clearing a Redo Log File

A redo log file might become corrupted while the database is open, and ultimately stop database activity because archiving cannot continue. In this situation the ALTER DATABASE CLEAR LOGFILE statement can be used reinitialize the file without shutting down the database.

The following statement clears the log files in redo log group number 3:

```
ALTER DATABASE CLEAR LOGFILE GROUP 3;
```

This statement overcomes two situations where dropping redo logs is not possible:

- If there are only two log groups
- The corrupt redo log file belongs to the current group

If the corrupt redo log file has not been archived, use the UNARCHIVED keyword in the statement.

```
ALTER DATABASE CLEAR UNARCHIVED LOGFILE GROUP 3;
```

This statement clears the corrupted redo logs and avoids archiving them. The cleared redo logs are available for use even though they were not archived.

If you clear a log file that is needed for recovery of a backup, then you can no longer recover from that backup. The database writes a message in the alert log describing the backups from which you cannot recover.

Note: If you clear an unarchived redo log file, you should make another backup of the database.

If you want to clear an unarchived redo log that is needed to bring an offline tablespace online, use the UNRECOVERABLE DATAFILE clause in the ALTER DATABASE CLEAR LOGFILE statement.

If you clear a redo log needed to bring an offline tablespace online, you will not be able to bring the tablespace online again. You will have to drop the tablespace or perform an incomplete recovery. Note that tablespaces taken offline normal do not require recovery.

Viewing Redo Log Information

The following views provide information on redo logs.

View	Description
V$LOG	Displays the redo log file information from the control file
V$LOGFILE	Identifies redo log groups and members and member status
V$LOG_HISTORY	Contains log history information

The following query returns the control file information about the redo log for a database.

```
SELECT * FROM V$LOG;

GROUP# THREAD#   SEQ   BYTES  MEMBERS  ARC STATUS     FIRST_CHANGE# FIRST_TIM
------ ------- ----- -------  -------  --- ---------  ------------- ---------
     1       1 10605 1048576        1  YES ACTIVE          11515628 16-APR-00
     2       1 10606 1048576        1  NO  CURRENT         11517595 16-APR-00
     3       1 10603 1048576        1  YES INACTIVE        11511666 16-APR-00
     4       1 10604 1048576        1  YES INACTIVE        11513647 16-APR-00
```

To see the names of all of the member of a group, use a query similar to the following:

```
SELECT * FROM V$LOGFILE;

GROUP#   STATUS  MEMBER
------  -------  --------------------------------
     1           D:\ORANT\ORADATA\IDDB2\REDO04.LOG
     2           D:\ORANT\ORADATA\IDDB2\REDO03.LOG
     3           D:\ORANT\ORADATA\IDDB2\REDO02.LOG
     4           D:\ORANT\ORADATA\IDDB2\REDO01.LOG
```

If STATUS is blank for a member, then the file is in use.

> **See Also:** *Oracle Database Reference* for detailed information about these views

7

Managing Archived Redo Logs

This chapter describes how to archive redo data. It contains the following topics:

- What Is the Archived Redo Log?
- Choosing Between NOARCHIVELOG and ARCHIVELOG Mode
- Controlling Archiving
- Specifying the Archive Destination
- Specifying the Mode of Log Transmission
- Managing Archive Destination Failure
- Controlling Trace Output Generated by the Archivelog Process
- Viewing Information About the Archived Redo Log

See Also:

- Part III, "Automated File and Storage Management" for information about creating an archived redo log that is both created and managed by the Oracle Database server
- *Oracle Real Application Clusters Administrator's Guide* for information specific to archiving in the Oracle Real Application Clusters environment

What Is the Archived Redo Log?

Oracle Database lets you save filled groups of redo log files to one or more offline destinations, known collectively as the **archived redo log**, or more simply the archive log. The process of turning redo log files into archived redo log files is called

archiving. This process is only possible if the database is running in **ARCHIVELOG mode**. You can choose automatic or manual archiving.

An archived redo log file is a copy of one of the filled members of a redo log group. It includes the redo entries and the unique log sequence number of the identical member of the redo log group. For example, if you are multiplexing your redo log, and if group 1 contains identical member files `a_log1` and `b_log1`, then the archiver process (ARC*n*) will archive one of these member files. Should `a_log1` become corrupted, then ARC*n* can still archive the identical `b_log1`. The archived redo log contains a copy of every group created since you enabled archiving.

When the database is running in `ARCHIVELOG` mode, the log writer process (LGWR) cannot reuse and hence overwrite a redo log group until it has been archived. The background process ARC*n* automates archiving operations when automatic archiving is enabled. The database starts multiple archiver processes as needed to ensure that the archiving of filled redo logs does not fall behind.

You can use archived redo logs to:

- Recover a database
- Update a standby database
- Get information about the history of a database using the LogMiner utility

> **See Also:** The following sources document the uses for archived redo logs:
>
> - *Oracle Database Backup and Recovery Basics*
> - *Oracle Data Guard Concepts and Administration* discusses setting up and maintaining a standby database
> - *Oracle Database Utilities* contains instructions for using the LogMiner PL/SQL package

Choosing Between NOARCHIVELOG and ARCHIVELOG Mode

This section describes the issues you must consider when choosing to run your database in `NOARCHIVELOG` or `ARCHIVELOG` mode, and contains these topics:

- Running a Database in NOARCHIVELOG Mode
- Running a Database in ARCHIVELOG Mode

The choice of whether to enable the archiving of filled groups of redo log files depends on the availability and reliability requirements of the application running

on the database. If you cannot afford to lose any data in your database in the event of a disk failure, use ARCHIVELOG mode. The archiving of filled redo log files can require you to perform extra administrative operations.

Running a Database in NOARCHIVELOG Mode

When you run your database in NOARCHIVELOG mode, you disable the archiving of the redo log. The database control file indicates that filled groups are not required to be archived. Therefore, when a filled group becomes inactive after a log switch, the group is available for reuse by LGWR.

NOARCHIVELOG mode protects a database from instance failure but not from media failure. Only the most recent changes made to the database, which are stored in the online redo log groups, are available for instance recovery. If a media failure occurs while the database is in NOARCHIVELOG mode, you can only restore the database to the point of the most recent full database backup. You cannot recover transactions subsequent to that backup.

In NOARCHIVELOG mode you cannot perform online tablespace backups, nor can you use online tablespace backups taken earlier while the database was in ARCHIVELOG mode. To restore a database operating in NOARCHIVELOG mode, you can use only whole database backups taken while the database is closed. Therefore, if you decide to operate a database in NOARCHIVELOG mode, take whole database backups at regular, frequent intervals.

Running a Database in ARCHIVELOG Mode

When you run a database in ARCHIVELOG mode, you enable the archiving of the redo log. The database control file indicates that a group of filled redo log files cannot be reused by LGWR until the group is archived. A filled group becomes available for archiving immediately after a redo log switch occurs.

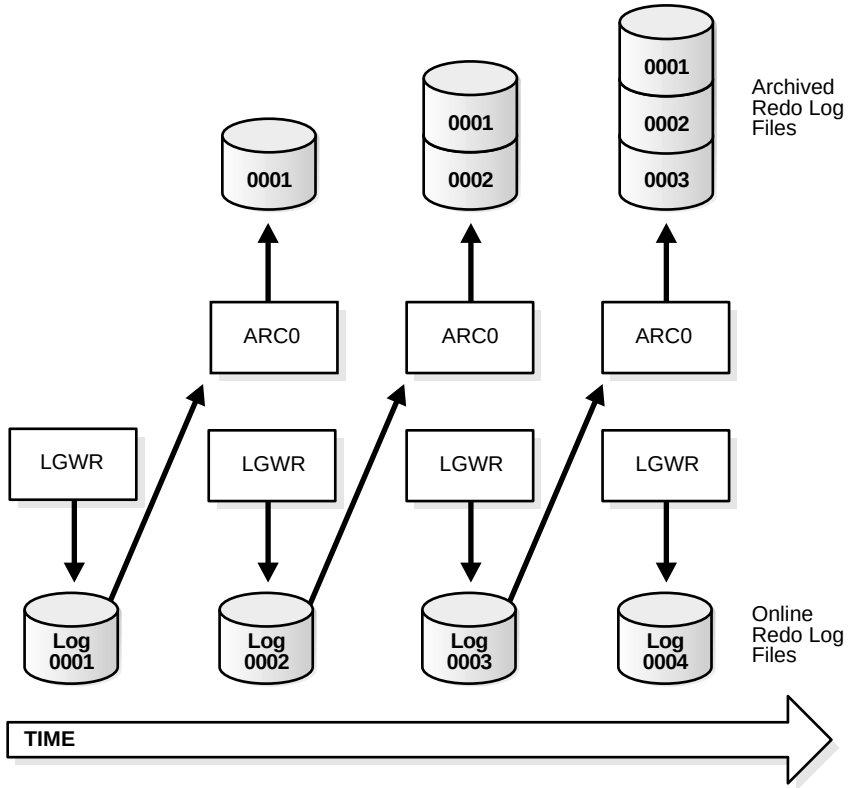
The archiving of filled groups has these advantages:

- A database backup, together with online and archived redo log files, guarantees that you can recover all committed transactions in the event of an operating system or disk failure.
- If you keep an archived log, you can use a backup taken while the database is open and in normal system use.
- You can keep a standby database current with its original database by continuously applying the original archived redo logs to the standby.

You can configure an instance to archive filled redo log files automatically, or you can archive manually. For convenience and efficiency, automatic archiving is usually best. Figure 7–1 illustrates how the archiver process (ARC0 in this illustration) writes filled redo log files to the database archived redo log.

If all databases in a distributed database operate in ARCHIVELOG mode, you can perform coordinated distributed database recovery. However, if any database in a distributed database is in NOARCHIVELOG mode, recovery of a global distributed database (to make all databases consistent) is limited by the last full backup of any database operating in NOARCHIVELOG mode.

Figure 7–1 Redo Log File Use in ARCHIVELOG Mode



Controlling Archiving

This section describes how to set the archiving mode of the database and how to control the archiving process. The following topics are discussed:

- Setting the Initial Database Archiving Mode
- Changing the Database Archiving Mode
- Performing Manual Archiving
- Adjusting the Number of Archiver Processes

> **See Also:** your Oracle operating system specific documentation for additional information on controlling archiving modes

Setting the Initial Database Archiving Mode

You set the initial archiving mode as part of database creation in the CREATE DATABASE statement. Usually, you can use the default of NOARCHIVELOG mode at database creation because there is no need to archive the redo information generated by that process. After creating the database, decide whether to change the initial archiving mode.

If you specify ARCHIVELOG mode, you must have initialization parameters set that specify the destinations for the archive log files (see "Specifying Archive Destinations" on page 7-8).

Changing the Database Archiving Mode

To change the archiving mode of the database, use the ALTER DATABASE statement with the ARCHIVELOG or NOARCHIVELOG clause. To change the archiving mode, you must be connected to the database with administrator privileges (AS SYSDBA).

The following steps switch the database archiving mode from NOARCHIVELOG to ARCHIVELOG:

1. Shut down the database instance.

   ```
   SHUTDOWN
   ```

   An open database must first be closed and any associated instances shut down before you can switch the database archiving mode. You cannot change the mode from ARCHIVELOG to NOARCHIVELOG if any datafiles need media recovery.

2. Back up the database.

   Before making any major change to a database, always back up the database to protect against any problems. This will be your final backup of the database in NOARCHIVELOG mode and can be used if something goes wrong during the change to ARCHIVELOG mode. See *Oracle Database Backup and Recovery Basics* for information about taking database backups.

3. Edit the initialization parameter file to include the initialization parameters that specify the destinations for the archive log files (see "Specifying Archive Destinations" on page 7-8).

4. Start a new instance and mount, but do not open, the database.

   ```
   STARTUP MOUNT
   ```

   To enable or disable archiving, the database must be mounted but not open.

5. Change the database archiving mode. Then open the database for normal operations.

   ```
   ALTER DATABASE ARCHIVELOG;
   ALTER DATABASE OPEN;
   ```

6. Shut down the database.

   ```
   SHUTDOWN IMMEDIATE
   ```

7. Back up the database.

   Changing the database archiving mode updates the control file. After changing the database archiving mode, you must back up all of your database files and control file. Any previous backup is no longer usable because it was taken in NOARCHIVELOG mode.

   > **See Also:** *Oracle Real Application Clusters Administrator's Guide* for more information about switching the archiving mode when using Real Application Clusters

Performing Manual Archiving

To operate your database in manual archiving mode, follow the procedure shown in "Changing the Database Archiving Mode" on page 7-5. However, when you specify the new mode in step 5, use the following statement:

```
ALTER DATABASE ARCHIVELOG MANUAL;
```

When you operate your database in manual ARCHIVELOG mode, you must archive inactive groups of filled redo log files or your database operation can be temporarily suspended. To archive a filled redo log group manually, connect with administrator privileges. Ensure that the database is mounted but not open. Use the ALTER SYSTEM statement with the ARCHIVE LOG clause to manually archive filled redo log files. The following statement archives all unarchived log files:

```
ALTER SYSTEM ARCHIVE LOG ALL;
```

When you use manual archiving mode, you cannot specify any standby databases in the archiving destinations.

Even when automatic archiving is enabled, you can use manual archiving for such actions as rearchiving an inactive group of filled redo log members to another location. In this case, it is possible for the instance to reuse the redo log group before you have finished manually archiving, and thereby overwrite the files. If this happens, the database writes an error message to the alert file.

Adjusting the Number of Archiver Processes

The LOG_ARCHIVE_MAX_PROCESSES initialization parameter specifies the number of ARC*n* processes that the database initially invokes. The default is two processes. There is usually no need specify this initialization parameter or to change its default value, because the database starts additional archiver processes (ARC*n*) as needed to ensure that the automatic processing of filled redo log files does not fall behind.

However, to avoid any runtime overhead of invoking additional ARC*n* processes, you can set the LOG_ARCHIVE_MAX_PROCESSES initialization parameter to specify up to ten ARC*n* processes to be started at instance startup. The LOG_ARCHIVE_MAX_PROCESSES parameter is dynamic, and can be changed using the ALTER SYSTEM statement. The database must be mounted but not open. The following statement increases (or decreases) the number of ARC*n* processes currently running:

```
ALTER SYSTEM SET LOG_ARCHIVE_MAX_PROCESSES=3;
```

Specifying the Archive Destination

Before you can archive redo logs, you must determine the destination to which you will archive and familiarize yourself with the various destination states. The dynamic performance (V$) views, listed in "Viewing Information About the Archived Redo Log" on page 7-19, provide all needed archive information.

The following topics are contained in this section:

Specifying Archive Destinations

You can choose whether to archive redo logs to a single destination or **multiplex** them. If you want to archive only to a single destination, you specify that destination in the LOG_ARCHIVE_DEST initialization parameter. If you want to multiplex the archived logs, you can choose whether to archive to up to ten locations (using the LOG_ARCHIVE_DEST_n parameters) or to archive only to a primary and secondary destination (using LOG_ARCHIVE_DEST and LOG_ARCHIVE_DUPLEX_DEST). The following table summarizes the multiplexing alternatives, which are further described in the sections that follow.

Method	Initialization Parameter	Host	Example
1	LOG_ARCHIVE_DEST_*n* where: *n* is an integer from 1 to 10	Local or remote	LOG_ARCHIVE_DEST_1 = 'LOCATION=/disk1/arc' LOG_ARCHIVE_DEST_2 = 'SERVICE=standby1'
2	LOG_ARCHIVE_DEST and LOG_ARCHIVE_DUPLEX_DEST	Local only	LOG_ARCHIVE_DEST = '/disk1/arc' LOG_ARCHIVE_DUPLEX_DEST = '/disk2/arc'

See Also:

- *Oracle Database Reference* for additional information about the initialization parameters used to control the archiving of redo logs
- *Oracle Data Guard Concepts and Administration* for information about using the LOG_ARCHIVE_DEST_*n* initialization parameter for specifying a standby destination. There are additional keywords that can be specified with this initialization parameter that are not discussed in this book.

Method 1: Using the LOG_ARCHIVE_DEST_*n* Parameter

Use the LOG_ARCHIVE_DEST_*n* parameter (where *n* is an integer from 1 to 10) to specify from one to ten different destinations for archival. Each numerically suffixed parameter uniquely identifies an individual destination.

You specify the location for LOG_ARCHIVE_DEST_*n* using the keywords explained in the following table:

Keyword	Indicates	Example
LOCATION	A local file system location.	LOG_ARCHIVE_DEST_1 = 'LOCATION=/disk1/arc'
SERVICE	Remote archival through Oracle Net service name.	LOG_ARCHIVE_DEST_2 = 'SERVICE=standby1'

If you use the LOCATION keyword, specify a valid path name for your operating system. If you specify SERVICE, the database translates the net service name through the tnsnames.ora file to a connect descriptor. The descriptor contains the information necessary for connecting to the remote database. The service name must have an associated database SID, so that the database correctly updates the log history of the control file for the standby database.

Perform the following steps to set the destination for archived redo logs using the LOG_ARCHIVE_DEST_*n* initialization parameter:

1. Use SQL*Plus to shut down the database.

   ```
   SHUTDOWN
   ```

2. Set the LOG_ARCHIVE_DEST_*n* initialization parameter to specify from one to ten archiving locations. The LOCATION keyword specifies an operating system specific path name. For example, enter:

   ```
   LOG_ARCHIVE_DEST_1 = 'LOCATION = /disk1/archive'
   LOG_ARCHIVE_DEST_2 = 'LOCATION = /disk2/archive'
   LOG_ARCHIVE_DEST_3 = 'LOCATION = /disk3/archive'
   ```

   If you are archiving to a standby database, use the SERVICE keyword to specify a valid net service name from the tnsnames.ora file. For example, enter:

   ```
   LOG_ARCHIVE_DEST_4 = 'SERVICE = standby1'
   ```

3. Optionally, set the LOG_ARCHIVE_FORMAT initialization parameter, using %t to include the thread number as part of the file name, %s to include the log sequence number, and %r to include the resetlogs ID (a timestamp value represented in ub4). Use capital letters (%T, %S, and %R) to pad the file name to the left with zeroes.

Note: If the COMPATIBLE initialization parameter is set to 10.0 or higher, the database requires the specification of resetlogs ID (%r) when you include the LOG_ARCHIVE_FORMAT parameter. The default for this parameter is operating system dependent. For example, this is the default format for UNIX:

```
LOG_ARCHIVE_FORMAT=%t_%s_%r.dbf
```

The incarnation of a database changes when you open it with the RESETLOGS option. Specifying %r causes the database to capture the resetlogs ID in the archive log file name, enabling you to more easily perform recovery from a backup of a previous database incarnation. See *Oracle Database Backup and Recovery Advanced User's Guide* for more information about this method of recovery.

The following example shows a setting of LOG_ARCHIVE_FORMAT:

```
LOG_ARCHIVE_FORMAT = arch_%t_%s_%r.arc
```

This setting will generate archived logs as follows for thread 1; log sequence numbers 100, 101, and 102; resetlogs ID 509210197. The identical resetlogs ID indicates that the files are all from the same database incarnation:

```
/disk1/archive/arch_1_100_509210197.arc,
/disk1/archive/arch_1_101_509210197.arc,
/disk1/archive/arch_1_102_509210197.arc

/disk2/archive/arch_1_100_509210197.arc,
/disk2/archive/arch_1_101_509210197.arc,
/disk2/archive/arch_1_102_509210197.arc

/disk3/archive/arch_1_100_509210197.arc,
/disk3/archive/arch_1_101_509210197.arc,
/disk3/archive/arch_1_102_509210197.arc
```

Method 2: Using LOG_ARCHIVE_DEST and LOG_ARCHIVE_DUPLEX_DEST

To specify a maximum of two locations, use the LOG_ARCHIVE_DEST parameter to specify a primary archive destination and the LOG_ARCHIVE_DUPLEX_DEST to specify an optional secondary archive destination. All locations must be local. Whenever the database archives a redo log, it archives it to every destination specified by either set of parameters.

Perform the following steps the use method 2:

1. Use SQL*Plus to shut down the database.

   ```
   SHUTDOWN
   ```

2. Specify destinations for the LOG_ARCHIVE_DEST and LOG_ARCHIVE_DUPLEX_DEST parameter (you can also specify LOG_ARCHIVE_DUPLEX_DEST dynamically using the ALTER SYSTEM statement). For example, enter:

   ```
   LOG_ARCHIVE_DEST = '/disk1/archive'
   LOG_ARCHIVE_DUPLEX_DEST = '/disk2/archive'
   ```

3. Set the LOG_ARCHIVE_FORMAT initialization parameter as described in step 3 for method 1.

Understanding Archive Destination Status

Each archive destination has the following variable characteristics that determine its status:

- **Valid/Invalid**: indicates whether the disk location or service name information is specified and valid
- **Enabled/Disabled**: indicates the availability state of the location and whether the database can use the destination
- **Active/Inactive**: indicates whether there was a problem accessing the destination

Several combinations of these characteristics are possible. To obtain the current status and other information about each destination for an instance, query the V$ARCHIVE_DEST view.

The characteristics determining a locations status that appear in the view are shown in Table 7–1. Note that for a destination to be used, its characteristics must be valid, enabled, and active.

Table 7–1 Destination Status

STATUS	Characteristics			Meaning
	Valid	Enabled	Active	
VALID	True	True	True	The user has properly initialized the destination, which is available for archiving.
INACTIVE	False	n/a	n/a	The user has not provided or has deleted the destination information.

Table 7–1 (Cont.) Destination Status

STATUS	Characteristics			Meaning
	Valid	Enabled	Active	
ERROR	True	True	False	An error occurred creating or writing to the destination file; refer to error data.
FULL	True	True	False	Destination is full (no disk space).
DEFERRED	True	False	True	The user manually and temporarily disabled the destination.
DISABLED	True	False	False	The user manually and temporarily disabled the destination following an error; refer to error data.
BAD PARAM	n/a	n/a	n/a	A parameter error occurred; refer to error data.

The LOG_ARCHIVE_DEST_STATE_*n* (where *n* is an integer from 1 to 10) initialization parameter lets you control the availability state of the specified destination (*n*).

- ENABLE indicates that the database can use the destination.
- DEFER indicates that the location is temporarily disabled.
- ALTERNATE indicates that the destination is an alternate.

The availability state of the destination is DEFER, unless there is a failure of its parent destination, in which case its state becomes ENABLE.

Specifying the Mode of Log Transmission

The two modes of transmitting archived logs to their destination are **normal archiving transmission** and **standby transmission** mode. Normal transmission involves transmitting files to a local disk. Standby transmission involves transmitting files through a network to either a local or remote standby database.

Normal Transmission Mode

In normal transmission mode, the archiving destination is another disk drive of the database server. In this configuration archiving does not contend with other files

required by the instance and can complete more quickly. Specify the destination with either the LOG_ARCHIVE_DEST_*n* or LOG_ARCHIVE_DEST parameters.

It is good practice to move archived redo log files and corresponding database backups from the local disk to permanent inexpensive offline storage media such as tape. A primary value of archived logs is database recovery, so you want to ensure that these logs are safe should disaster strike your primary database.

Standby Transmission Mode

In standby transmission mode, the archiving destination is either a local or remote standby database.

> **Caution:** You can maintain a standby database on a local disk, but Oracle strongly encourages you to maximize disaster protection by maintaining your standby database at a remote site.

If you are operating your standby database in **managed recovery mode**, you can keep your standby database synchronized with your source database by automatically applying transmitted archive logs.

To transmit files successfully to a standby database, either ARC*n* or a server process must do the following:

- Recognize a remote location
- Transmit the archived logs in conjunction with a **remote file server** (RFS) process that resides on the remote server

Each ARC*n* process has a corresponding RFS for each standby destination. For example, if three ARC*n* processes are archiving to two standby databases, then Oracle Database establishes six RFS connections.

You transmit archived logs through a network to a remote location by using Oracle Net Services. Indicate a remote archival by specifying a Oracle Net service name as an attribute of the destination. Oracle Database then translates the service name, through the tnsnames.ora file, to a connect descriptor. The descriptor contains the information necessary for connecting to the remote database. The service name must have an associated database SID, so that the database correctly updates the log history of the control file for the standby database.

The RFS process, which runs on the destination node, acts as a network server to the ARC*n* client. Essentially, ARC*n* pushes information to RFS, which transmits it to the standby database.

The RFS process, which is required when archiving to a remote destination, is responsible for the following tasks:

- Consuming network I/O from the ARC*n* process
- Creating file names on the standby database by using the `STANDBY_ARCHIVE_DEST` parameter
- Populating the log files at the remote site
- Updating the standby database control file (which Recovery Manager can then use for recovery)

Archived redo logs are integral to maintaining a standby database, which is an exact replica of a database. You can operate your database in standby archiving mode, which automatically updates a standby database with archived redo logs from the original database.

See Also:

- *Oracle Data Guard Concepts and Administration*
- *Oracle Net Services Administrator's Guide* for information about connecting to a remote database using a service name

Managing Archive Destination Failure

Sometimes archive destinations can fail, causing problems when you operate in automatic archiving mode. Oracle Database provides procedures to help you minimize the problems associated with destination failure. These procedures are discussed in the sections that follow:

- Specifying the Minimum Number of Successful Destinations
- Rearchiving to a Failed Destination

Specifying the Minimum Number of Successful Destinations

The optional initialization parameter `LOG_ARCHIVE_MIN_SUCCEED_DEST=`*n* determines the minimum number of destinations to which the database must successfully archive a redo log group before it can reuse online log files. The default

value is 1. Valid values for *n* are 1 to 2 if you are using duplexing, or 1 to 10 if you are multiplexing.

Specifying Mandatory and Optional Destinations

The LOG_ARCHIVE_DEST_*n* parameter lets you specify whether a destination is OPTIONAL (the default) or MANDATORY. The LOG_ARCHIVE_MIN_SUCCEED_DEST=*n* parameter uses all MANDATORY destinations plus some number of nonstandby OPTIONAL destinations to determine whether LGWR can overwrite the online log. The following rules apply:

- Omitting the MANDATORY attribute for a destination is the same as specifying OPTIONAL.
- You must have at least one local destination, which you can declare OPTIONAL or MANDATORY.
- When you specify a value for LOG_ARCHIVE_MIN_SUCCEED_DEST=*n*, Oracle Database will treat at least one local destination as MANDATORY, because the minimum value for LOG_ARCHIVE_MIN_SUCCEED_DEST is 1.
- If any MANDATORY destination fails, including a MANDATORY standby destination, Oracle Database ignores the LOG_ARCHIVE_MIN_SUCCEED_DEST parameter.
- The LOG_ARCHIVE_MIN_SUCCEED_DEST value cannot be greater than the number of destinations, nor can it be greater than the number of MANDATORY destinations plus the number of OPTIONAL local destinations.
- If you DEFER a MANDATORY destination, and the database overwrites the online log without transferring the archived log to the standby site, then you must transfer the log to the standby manually.

If you are duplexing the archived logs, you can establish which destinations are mandatory or optional by using the LOG_ARCHIVE_DEST and LOG_ARCHIVE_DUPLEX_DEST parameters. The following rules apply:

- Any destination declared by LOG_ARCHIVE_DEST is mandatory.
- Any destination declared by LOG_ARCHIVE_DUPLEX_DEST is optional if LOG_ARCHIVE_MIN_SUCCEED_DEST = 1 and mandatory if LOG_ARCHIVE_MIN_SUCCEED_DEST = 2.

Specifying the Number of Successful Destinations: Scenarios

You can see the relationship between the LOG_ARCHIVE_DEST_*n* and LOG_ARCHIVE_MIN_SUCCEED_DEST parameters most easily through sample scenarios.

Scenario for Archiving to Optional Local Destinations In this scenario, you archive to three local destinations, each of which you declare as OPTIONAL. Table 7–2 illustrates the possible values for LOG_ARCHIVE_MIN_SUCCEED_DEST=*n* in this case.

Table 7–2 LOG_ARCHIVE_MIN_SUCCEED_DEST Values for Scenario 1

Value	Meaning
1	The database can reuse log files only if at least one of the OPTIONAL destinations succeeds.
2	The database can reuse log files only if at least two of the OPTIONAL destinations succeed.
3	The database can reuse log files only if all of the OPTIONAL destinations succeed.
4 or greater	ERROR: The value is greater than the number of destinations.

This scenario shows that even though you do not explicitly set any of your destinations to MANDATORY using the LOG_ARCHIVE_DEST_*n* parameter, the database must successfully archive to one or more of these locations when LOG_ARCHIVE_MIN_SUCCEED_DEST is set to 1, 2, or 3.

Scenario for Archiving to Both Mandatory and Optional Destinations Consider a case in which:

- You specify two MANDATORY destinations.
- You specify two OPTIONAL destinations.
- No destination is a standby database.

Table 7–3 shows the possible values for LOG_ARCHIVE_MIN_SUCCEED_DEST=*n*.

Table 7–3 LOG_ARCHIVE_MIN_SUCCEED_DEST Values for Scenario 2

Value	Meaning
1	The database ignores the value and uses the number of MANDATORY destinations (in this example, 2).

Table 7–3 LOG_ARCHIVE_MIN_SUCCEED_DEST Values for Scenario 2

Value	Meaning
2	The database can reuse log files even if no OPTIONAL destination succeeds.
3	The database can reuse logs only if at least one OPTIONAL destination succeeds.
4	The database can reuse logs only if both OPTIONAL destinations succeed.
5 or greater	ERROR: The value is greater than the number of destinations.

This case shows that the database must archive to the destinations you specify as MANDATORY, regardless of whether you set LOG_ARCHIVE_MIN_SUCCEED_DEST to archive to a smaller number of destinations.

Rearchiving to a Failed Destination

Use the REOPEN attribute of the LOG_ARCHIVE_DEST_*n* parameter to specify whether and when ARC*n* should attempt to rearchive to a failed destination following an error. REOPEN applies to all errors, not just OPEN errors.

REOPEN=*n* sets the minimum number of seconds before ARC*n* should try to reopen a failed destination. The default value for *n* is 300 seconds. A value of 0 is the same as turning off the REOPEN attribute; ARC*n* will not attempt to archive after a failure. If you do not specify the REOPEN keyword, ARC*n* will never reopen a destination following an error.

You cannot use REOPEN to specify the number of attempts ARC*n* should make to reconnect and transfer archived logs. The REOPEN attempt either succeeds or fails.

When you specify REOPEN for an OPTIONAL destination, the database can overwrite online logs if there is an error. If you specify REOPEN for a MANDATORY destination, the database stalls the production database when it cannot successfully archive. In this situation, consider the following options:

- Archive manually to the failed destination.
- Change the destination by deferring the destination, specifying the destination as optional, or changing the service.
- Drop the destination.

When using the REOPEN keyword, note the following:

- ARC*n* reopens a destination only when *starting* an archive operation from the beginning of the log file, never *during* a current operation. ARC*n* always retries the log copy from the beginning.
- If you specified REOPEN, either with a specified time the default, ARC*n* checks to see whether the time of the recorded error plus the REOPEN interval is less than the current time. If it is, ARC*n* retries the log copy.
- The REOPEN clause successfully affects the ACTIVE=TRUE destination state. The VALID and ENABLED states are not changed.

Controlling Trace Output Generated by the Archivelog Process

Background processes always write to a trace file when appropriate. (See the discussion of this topic in "Monitoring the Database Using Trace Files and the Alert File" on page 4-29.) In the case of the archivelog process, you can control the output that is generated to the trace file. You do this by setting the LOG_ARCHIVE_TRACE initialization parameter to specify a **trace level**. The following values can be specified:

Trace Level	Meaning
0	Disable archivelog tracing. This is the default.
1	Track archival of redo log file.
2	Track archival status for each archivelog destination.
4	Track archival operational phase.
8	Track archivelog destination activity.
16	Track detailed archivelog destination activity.
32	Track archivelog destination parameter modifications.
64	Track ARC*n* process state activity.
128	Track FAL (fetch archived log) server related activities.
256	Supported in a future release.
512	Tracks asynchronous LGWR activity.
1024	RFS physical client tracking.
2048	ARC*n*/RFS heartbeat tracking.
4096	Track real-time apply

You can combine tracing levels by specifying a value equal to the sum of the individual levels that you would like to trace. For example, setting LOG_ARCHIVE_TRACE=12, will generate trace level 8 and 4 output. You can set different values for the primary and any standby database.

The default value for the LOG_ARCHIVE_TRACE parameter is 0. At this level, the archivelog process generates appropriate alert and trace entries for error conditions.

You can change the value of this parameter dynamically using the ALTER SYSTEM statement. The database must be mounted but not open. For example:

```
ALTER SYSTEM SET LOG_ARCHIVE_TRACE=12;
```

Changes initiated in this manner will take effect at the start of the next archiving operation.

> **See Also:** *Oracle Data Guard Concepts and Administration* for information about using this parameter with a standby database

Viewing Information About the Archived Redo Log

You can display information about the archived redo logs using the following sources:

- Dynamic Performance Views
- The ARCHIVE LOG LIST Command

Dynamic Performance Views

Several dynamic performance views contain useful information about archived redo logs, as summarized in the following table.

Dynamic Performance View	Description
V$DATABASE	Identifies whether the database is in ARCHIVELOG or NOARCHIVELOG mode and whether MANUAL (archiving mode) has been specified.
V$ARCHIVED_LOG	Displays historical archived log information from the control file. If you use a recovery catalog, the RC_ARCHIVED_LOG view contains similar information.
V$ARCHIVE_DEST	Describes the current instance, all archive destinations, and the current value, mode, and status of these destinations.

Dynamic Performance View	Description
V$ARCHIVE_PROCESSES	Displays information about the state of the various archive processes for an instance.
V$BACKUP_REDOLOG	Contains information about any backups of archived logs. If you use a recovery catalog, the RC_BACKUP_REDOLOG contains similar information.
V$LOG	Displays all redo log groups for the database and indicates which need to be archived.
V$LOG_HISTORY	Contains log history information such as which logs have been archived and the SCN range for each archived log.

For example, the following query displays which redo log group requires archiving:

```
SELECT GROUP#, ARCHIVED
   FROM SYS.V$LOG;

GROUP#     ARC
--------   ---
       1   YES
       2   NO
```

To see the current archiving mode, query the V$DATABASE view:

```
SELECT LOG_MODE FROM SYS.V$DATABASE;

LOG_MODE
------------
NOARCHIVELOG
```

See Also: *Oracle Database Reference* for detailed descriptions of dynamic performance views

The ARCHIVE LOG LIST Command

The SQL*Plus command ARCHIVE LOG LIST displays archiving information for the connected instance. For example:

```
SQL> ARCHIVE LOG LIST

Database log mode              Archive Mode
Automatic archival             Enabled
Archive destination            D:\oracle\oradata\IDDB2\archive
```

```
Oldest online log sequence     11160
Next log sequence to archive   11163
Current log sequence           11163
```

This display tells you all the necessary information regarding the archived redo log settings for the current instance:

- The database is currently operating in ARCHIVELOG mode.
- Automatic archiving is enabled.
- The archived redo log destination is D:\oracle\oradata\IDDB2\archive .
- The oldest filled redo log group has a sequence number of 11160.
- The next filled redo log group to archive has a sequence number of 11163.
- The current redo log file has a sequence number of 11163.

> **See Also:** *SQL*Plus User's Guide and Reference* for more information on the ARCHIVE LOG LIST command

8

Managing Tablespaces

This chapter describes the various aspects of tablespace management, and contains the following topics:

- Guidelines for Managing Tablespaces
- Creating Tablespaces
- Coalescing Free Space in Dictionary-Managed Tablespaces
- Specifying Nonstandard Block Sizes for Tablespaces
- Controlling the Writing of Redo Records
- Altering Tablespace Availability
- Using Read-Only Tablespaces
- Renaming Tablespaces
- Dropping Tablespaces
- Managing the SYSAUX Tablespace
- Diagnosing and Repairing Locally Managed Tablespace Problems
- Migrating the SYSTEM Tablespace to a Locally Managed Tablespace
- Transporting Tablespaces Between Databases
- Viewing Tablespace Information

> **See Also:** Chapter 11, "Using Oracle-Managed Files" for information about creating datafiles and tempfiles that are both created and managed by the Oracle Database server

Guidelines for Managing Tablespaces

Before working with tablespaces of an Oracle Database, familiarize yourself with the guidelines provided in the following sections:

- Using Multiple Tablespaces
- Assigning Tablespace Quotas to Users

> **See Also:** *Oracle Database Concepts* for a complete discussion of database structure, space management, tablespaces, and datafiles

Using Multiple Tablespaces

Using multiple tablespaces allows you more flexibility in performing database operations. When a database has multiple tablespaces, you can:

- Separate user data from data dictionary data to reduce contention among dictionary objects and schema objects for the same datafiles.
- Separate data of one application from the data of another to prevent multiple applications from being affected if a tablespace must be taken offline.
- Store different the datafiles of different tablespaces on different disk drives to reduce I/O contention.
- Take individual tablespaces offline while others remain online, providing better overall availability.
- Optimizing tablespace use by reserving a tablespace for a particular type of database use, such as high update activity, read-only activity, or temporary segment storage.
- Back up individual tablespaces.

Some operating systems set a limit on the number of files that can be open simultaneously. Such limits can affect the number of tablespaces that can be simultaneously online. To avoid exceeding your operating system limit, plan your tablespaces efficiently. Create only enough tablespaces to fill your needs, and create these tablespaces with as few files as possible. If you need to increase the size of a tablespace, add one or two large datafiles, or create datafiles with autoextension enabled, rather than creating many small datafiles.

Review your data in light of these factors and decide how many tablespaces you need for your database design.

Assigning Tablespace Quotas to Users

Grant to users who will be creating tables, clusters, materialized views, indexes, and other objects the privilege to create the object and a **quota** (space allowance or limit) in the tablespace intended to hold the object segment.

> **See Also:** *Oracle Database Security Guide* for information about creating users and assigning tablespace quotas.

Creating Tablespaces

Before you can create a tablespace, you must create a database to contain it. The primary tablespace in any database is the SYSTEM tablespace, which contains information basic to the functioning of the database server, such as the data dictionary and the system rollback segment. The SYSTEM tablespace is the first tablespace created at database creation. It is managed as any other tablespace, but requires a higher level of privilege and is restricted in some ways. For example, you cannot rename or drop the SYSTEM tablespace or take it offline.

The SYSAUX tablespace, which acts as an auxiliary tablespace to the SYSTEM tablespace, is also always created when you create a database. It contains information about and the schemas used by various Oracle products and features, so that those products do not require their own tablespaces. As for the SYSTEM tablespace, management of the SYSAUX tablespace requires a higher level of security and you cannot rename or drop it. The management of the SYSAUX tablespace is discussed separately in "Managing the SYSAUX Tablespace" on page 8-34.

The steps for creating tablespaces vary by operating system, but the first step is always to use your operating system to create a directory structure in which your datafiles will be allocated. On most operating systems, you specify the size and fully specified filenames of datafiles when you create a new tablespace or alter an existing tablespace by adding datafiles. Whether you are creating a new tablespace or modifying an existing one, the database automatically allocates and formats the datafiles as specified.

To create a new tablespace, use the SQL statement CREATE TABLESPACE or CREATE TEMPORARY TABLESPACE. You must have the CREATE TABLESPACE system privilege to create a tablespace. Later, you can use the ALTER TABLESPACE or ALTER DATABASE statements to alter the tablespace. You must have the ALTER TABLESPACE or ALTER DATABASE system privilege, correspondingly.

You can also use the CREATE UNDO TABLESPACE statement to create a special type of tablespace called an **undo tablespace**, which is specifically designed to contain

undo records. These are records generated by the database that are used to roll back, or undo, changes to the database for recovery, read consistency, or as requested by a ROLLBACK statement. Creating and managing undo tablespaces is the subject of Chapter 10, "Managing the Undo Tablespace".

The creation and maintenance of permanent and temporary tablespaces are discussed in the following sections:

- Locally Managed Tablespaces
- Bigfile Tablespaces
- Dictionary-Managed Tablespaces
- Temporary Tablespaces
- Multiple Temporary Tablespaces: Using Tablespace Groups

See Also:

- Chapter 2, "Creating an Oracle Database" and your Oracle Database installation documentation for your operating system for information about tablespaces that are created at database creation
- *Oracle Database SQL Reference* for more information about the syntax and semantics of the CREATE TABLESPACE, CREATE TEMPORARY TABLESPACE, ALTER TABLESPACE, and ALTER DATABASE statements.
- "Specifying Database Block Sizes" on page 2-31 for information about initialization parameters necessary to create tablespaces with nonstandard block sizes

Locally Managed Tablespaces

Locally managed tablespaces track all extent information in the tablespace itself by using bitmaps, resulting in the following benefits:

- Concurrency and speed of space operations is improved, because space allocations and deallocations modify locally managed resources (bitmaps stored in header files) rather than requiring centrally managed resources such as enqueues
- Performance is improved, because recursive operations that are sometimes required during dictionary-managed space allocation are eliminated

- Readable standby databases are allowed, because locally managed temporary tablespaces (used, for example, for sorts) are locally managed and thus do not generate any undo or redo.
- Space allocation is simplified, because when the AUTOALLOCATE clause is specified, the database automatically selects the appropriate extent size.
- User reliance on the data dictionary is reduced, because the necessary information is stored in file headers and bitmap blocks.
- Coalescing free extents is unnecessary for locally managed tablespaces.

All tablespaces, including the SYSTEM tablespace, can be locally managed.

The DBMS_SPACE_ADMIN package provides maintenance procedures for locally managed tablespaces.

See Also:

- "Creating a Locally Managed SYSTEM Tablespace" on page 2-15, "Migrating the SYSTEM Tablespace to a Locally Managed Tablespace" on page 8-40, and "Diagnosing and Repairing Locally Managed Tablespace Problems" on page 8-36
- "Bigfile Tablespaces" on page 8-9 for information about creating another type of locally managed tablespace that contains only a single datafile or tempfile.
- *PL/SQL Packages and Types Reference* for information on the DBMS_SPACE_ADMIN package

Creating a Locally Managed Tablespace

You create a locally managed tablespace by specifying LOCAL in the EXTENT MANAGEMENT clause of the CREATE TABLESPACE statement. This is the default for new permanent tablespaces, but you must specify if it you want to specify the management of the locally managed tablespace. You can have the database manage extents for you automatically with the AUTOALLOCATE clause (the default), or you can specify that the tablespace is managed with uniform extents of a specific size (UNIFORM).

If you expect the tablespace to contain objects of varying sizes requiring many extents with different extent sizes, then AUTOALLOCATE is the best choice. AUTOALLOCATE is also a good choice if it is not important for you to have a lot of control over space allocation and deallocation, because it simplifies tablespace

management. Some space may be wasted with this setting, but the benefit of having Oracle Database manage your space most likely outweighs this drawback.

If you want exact control over unused space, and you can predict exactly the space to be allocated for an object or objects and the number and size of extents, then UNIFORM is a good choice. This setting ensures that you will never have unusable space in your tablespace.

When you do not explicitly specify the type of extent management, Oracle Database determines extent management as follows:

- If the CREATE TABLESPACE statement omits the DEFAULT storage clause, then the database creates a locally managed autoallocated tablespace.
- If the CREATE TABLESPACE statement includes a DEFAULT storage clause, then the database considers the following:
  - If you specified the MINIMUM EXTENT clause, the database evaluates whether the values of MINIMUM EXTENT, INITIAL, and NEXT are equal and the value of PCTINCREASE is 0. If so, the database creates a locally managed uniform tablespace with extent size = INITIAL. If the MINIMUM EXTENT, INITIAL, and NEXT parameters are not equal, or if PCTINCREASE is not 0, the database ignores any extent storage parameters you may specify and creates a locally managed, autoallocated tablespace.
  - If you did not specify MINIMUM EXTENT clause, the database evaluates only whether the storage values of INITIAL and NEXT are equal and PCTINCREASE is 0. If so, the tablespace is locally managed and uniform. Otherwise, the tablespace is locally managed and autoallocated.

The following statement creates a locally managed tablespace named lmtbsb and specifies AUTOALLOCATE:

```
CREATE TABLESPACE lmtbsb DATAFILE '/u02/oracle/data/lmtbsb01.dbf' SIZE 50M
    EXTENT MANAGEMENT LOCAL AUTOALLOCATE;
```

AUTOALLOCATE causes the tablespace to be system managed with a minimum extent size of 64K. In contrast, dictionary-managed tablespaces have a minimum extent size of two database blocks. Therefore, in systems with block size smaller than 32K, autoallocated locally managed tablespace will be larger initially.

The alternative to AUTOALLOCATE is UNIFORM. which specifies that the tablespace is managed with extents of uniform size. You can specify that size in the SIZE clause of UNIFORM. If you omit SIZE, then the default size is 1M.

The following example creates a tablespace with uniform 128K extents. (In a database with 2K blocks, each extent would be equivalent to 64 database blocks). Each 128K extent is represented by a bit in the extent bitmap for this file.

```
CREATE TABLESPACE lmtbsb DATAFILE '/u02/oracle/data/lmtbsb01.dbf' SIZE 50M
    EXTENT MANAGEMENT LOCAL UNIFORM SIZE 128K;
```

You cannot specify the DEFAULT storage clause, MINIMUM EXTENT, or TEMPORARY when you explicitly specify EXTENT MANAGEMENT LOCAL. If you want to create a temporary locally managed tablespace, use the CREATE TEMPORARY TABLESPACE statement.

Note: When you allocate a datafile for a locally managed tablespace, you should allow space for metadata used for space management (the extent bitmap or space header segment) which are part of user space. For example, if specify the UNIFORM clause in the extent management clause but you omit the SIZE parameter, then the default extent size is 1MB. In that case, the size specified for the datafile must be larger (at least one block plus space for the bitmap) than 1MB.

Specifying Segment Space Management in Locally Managed Tablespaces

When you create a locally managed tablespace using the CREATE TABLESPACE statement, the SEGMENT SPACE MANAGEMENT clause lets you specify how free and used space within a segment is to be managed. You can choose either manual or automatic segment-space management.

- MANUAL: Manual segment-space management uses free lists to manage free space within segments. Free lists are lists of data blocks that have space available for inserting rows. With this form of segment-space management, you must specify and tune the PCTUSED, FREELISTS, and FREELIST GROUPS storage parameters for schema objects created in the tablespace. MANUAL is the default.
- AUTO: Automatic segment-space management uses bitmaps to manage the free space within segments. The bitmap describes the status of each data block within a segment with respect to the amount of space in the block available for inserting rows. As more or less space becomes available in a data block, its new state is reflected in the bitmap. These bitmaps allow the database to manage free space automatically.

You can specify automatic segment-space management only for permanent, locally managed tablespaces. Automatic segment-space management is a simpler and more efficient way of managing space within a segment. It completely eliminates any need to specify and tune the PCTUSED, FREELISTS, and FREELIST GROUPS storage parameters for schema objects created in the tablespace. If you specify these attributes, the database ignores them.

Automatic segment-space management delivers better space utilization than manual segment-space management. It is also self-tuning, in that it scales with increasing number of users or instances. In a Real Application Clusters environment, automatic segment-space management allows for a dynamic affinity of space to instances, thus avoiding the hard partitioning of space inherent with using free list groups. In addition, for many standard workloads, application performance with automatic segment-space management is better than the performance of a well-tuned application using manual segment-space management.

The following statement creates tablespace lmtbsb with automatic segment-space management:

```
CREATE TABLESPACE lmtbsb DATAFILE '/u02/oracle/data/lmtbsb01.dbf' SIZE 50M
    EXTENT MANAGEMENT LOCAL
    SEGMENT SPACE MANAGEMENT AUTO;
```

The segment-space management you specify at tablespace creation time applies to all segments subsequently created in the tablespace. You cannot subsequently change the segment-space management mode of a tablespace.

Notes: If you specify AUTO segment management, then:

- If you set extent management to LOCAL UNIFORM, then you must ensure that each extent contains at least 5 database blocks.
- If you set extent management to LOCAL AUTOALLOCATE, and if the database block size is 16K or greater, then Oracle manages segment space by creating extents with a minimum size of 5 blocks rounded up to 64K.

Locally managed tablespaces using automatic segment-space management can be created as single-file, or bigfile, tablespaces, as described in "Bigfile Tablespaces" on page 8-9.

Altering a Locally Managed Tablespace

You cannot alter a locally managed tablespace to a locally managed temporary tablespace, nor can you change its method of segment-space management. Coalescing free extents is unnecessary for locally managed tablespaces. However, you can use the ALTER TABLESPACE statement on locally managed tablespaces for some operations, including the following:

- Adding a datafile. For example:

```
ALTER TABLESPACE lmtbsb
   ADD DATAFILE '/u02/oracle/data/lmtbsb02.dbf' SIZE 1M;
```

- Altering tablespace availability (ONLINE / OFFLINE). See "Altering Tablespace Availability" on page 8-25.
- Making a tablespace read-only or read / write. See "Using Read-Only Tablespaces" on page 8-27.
- Renaming a datafile, or enabling or disabling the autoextension of the size of a datafile in the tablespace. See Chapter 9, "Managing Datafiles and Tempfiles".

Bigfile Tablespaces

A **bigfile tablespace** is a tablespace with a single, but very large (up to 4G blocks) datafile. Traditional smallfile tablespaces, in contrast, can contain multiple datafiles, but the files cannot be as large. The benefits of bigfile tablespaces are the following:

- A bigfile tablespace with 8K blocks can contain a 32 terabyte datafile. A bigfile tablespace with 32K blocks can contain a 128 terabyte datafile. The maximum number of datafiles in an Oracle Database is limited (usually to 64K files). Therefore, bigfile tablespaces can significantly enhance the storage capacity of an Oracle Database.
- Bigfile tablespaces can reduce the number of datafiles needed for a database. An additional benefit is that the DB_FILES initialization parameter and MAXDATAFILES parameter of the CREATE DATABASE and CREATE CONTROLFILE statements can be adjusted to reduce the amount of SGA space required for datafile information and the size of the control file.
- Bigfile tablespaces simplify database management by providing datafile transparency. SQL syntax for the ALTER TABLESPACE statement lets you perform operations on tablespaces, rather than the underlying individual datafiles.

Bigfile tablespaces are supported only for locally managed tablespaces with automatic segment-space management, with three exceptions: locally managed undo tablespaces, temporary tablespaces, and the SYSTEM tablespace can be bigfile tablespaces even if their segments are manually managed.

Notes:

- Bigfile tablespaces are intended to be used with Automatic Storage Management (ASM) or other logical volume managers that supports striping or RAID, and dynamically extensible logical volumes.
- Avoid creating bigfile tablespaces on a system that does not support striping because of negative implications for parallel query execution and RMAN backup parallelization.
- Using bigfile tablespaces on platforms that do not support large file sizes is not recommended and can limit tablespace capacity. Refer to your operating system specific documentation for information about maximum supported file sizes.

Creating a Bigfile Tablespace

To create a bigfile tablespace, specify the BIGFILE keyword of the CREATE TABLESPACE statement (CREATE BIGFILE TABLESPACE ...). Oracle Database automatically creates a locally managed tablespace with automatic segment-spec management. You can, but need not, specify EXTENT MANAGEMENT LOCAL and SEGMENT SPACE MANAGEMENT AUTO in this statement. However, the database returns an error if you specify EXTENT MANAGEMENT DICTIONARY or SEGMENT SPACE MANAGEMENT MANUAL. The remaining syntax of the statement is the same as for the CREATE TABLESPACE statement, but you can only specify one datafile. For example:

```
CREATE BIGFILE TABLESPACE bigtbs
    DATAFILE '/u02/oracle/data/bigtbs01.dbf' SIZE 50G
...
```

You can specify SIZE in kilobytes (K), megabytes (M), gigabytes (G), or terabytes (T).

If the default tablespace type was set to BIGFILE at database creation, you need not specify the keyword BIGFILE in the CREATE TABLESPACE statement. A bigfile tablespace is created by default.

If the default tablespace type was set to BIGFILE at database creation, but you want to create a traditional (smallfile) tablespace, then specify a CREATE SMALLFILE TABLESPACE statement to override the default tablespace type for the tablespace that you are creating.

> **See Also:** "Supporting Bigfile Tablespaces During Database Creation" on page 2-23

Altering a Bigfile Tablespace

Two clauses of the ALTER TABLESPACE statement support datafile transparency when you are using bigfile tablespaces:

- RESIZE: The RESIZE clause lets you resize the single datafile in a bigfile tablespace to an absolute size, without referring to the datafile. For example:

  ```
  ALTER TABLESPACE bigtbs RESIZE 80G;
  ```

- AUTOEXTEND (used outside of the ADD DATAFILE clause):

  With a bigfile tablespace, you can use the AUTOEXTEND clause outside of the ADD DATAFILE clause. For example:

  ```
  ALTER TABLESPACE bigtbs AUTOEXTEND ON NEXT 20G;
  ```

An error is raised if you specify an ADD DATAFILE clause for a bigfile tablespace.

Identifying a Bigfile Tablespace

The following views contain a BIGFILE column that identifies a tablespace as a bigfile tablespace:

- DBA_TABLESPACES
- USER_TABLESPACES
- V$TABLESPACE

You can also identify a bigfile tablespace by the relative file number of its single datafile. That number is 1024 on most platforms, but 4096 on OS/390.

Dictionary-Managed Tablespaces

The default for extent management when creating a tablespace is locally managed. However, you can explicitly specify a dictionary-managed tablespace. For dictionary-managed tablespaces, the database updates the appropriate tables in the data dictionary whenever an extent is allocated or freed for reuse.

Creating a Dictionary-Managed Tablespace

The following statement creates the dictionary-managed tablespace `tbsa`:

```
CREATE TABLESPACE tbsa
    DATAFILE '/u02/oracle/data/tbsa01.dbf' SIZE 50M
    EXTENT MANAGEMENT DICTIONARY
    DEFAULT STORAGE (
        INITIAL 50K
        NEXT 50K
        MINEXTENTS 2
        MAXEXTENTS 50
        PCTINCREASE 0);
```

The tablespace has the following characteristics:

- The data of the new tablespace is contained in a single datafile, 50M in size.
- The tablespace is explicitly created as a dictionary-managed tablespace by specifying `EXTENT MANAGEMENT DICTIONARY`.
- The default storage parameters for any segments created in this tablespace are specified.

The parameters specified in the preceding example determine segment storage allocation in the tablespace. These parameters affect both how long it takes to access data stored in the database and how efficiently space in the database is used. They are referred to as storage parameters, and are described in the following table:

Storage Parameter	Description
`INITIAL`	Defines the size in bytes of the first extent in the segment
`NEXT`	Defines the size in bytes of the second and subsequent extents
`PCTINCREASE`	Specifies the percent by which each extent, after the second (`NEXT`) extent, grows
`MINEXTENTS`	Specifies the number of extents allocated when a segment is first created in the tablespace
`MAXEXTENTS`	Specifies the maximum number of extents that a segment can have. Can also be specified as `UNLIMITED`.

The `MINIMUM EXTENT` parameter on the `CREATE TABLESPACE` statement also influences segment allocation. If specified, it ensures that all free and allocated extents in the tablespace are at least as large as, and a multiple of, a specified

number of bytes. This clause provides one way to control free space fragmentation in the tablespace.

See Also:

- "Managing Storage Parameters" on page 13-10
- *Oracle Database SQL Reference* for a complete description of storage parameters

Specifying Tablespace Default Storage Parameters

When you create a new dictionary-managed tablespace, you can specify default storage parameters for objects that will be created in the tablespace. Storage parameters specified when an object is created override the default storage parameters of the tablespace containing the object. If you do not specify storage parameters when creating an object, the object segment automatically uses the default storage parameters for the tablespace.

Set the default storage parameters for a tablespace to account for the size of a typical object that the tablespace will contain (you estimate this size). You can specify different storage parameters for an unusual or exceptional object when creating that object. You can also alter your default storage parameters at a later time.

You cannot specify default storage parameters for tablespaces that are specifically created as locally managed.

Note: If you do not specify the default storage parameters for a new dictionary-managed tablespace, Oracle Database chooses default storage parameters appropriate for your operating system.

Altering a Dictionary-Managed Tablespace

One common change to a database is adding a datafile. The following statement creates a new datafile for the `tbsa` tablespace:

```
ALTER TABLESPACE tbsa
   ADD DATAFILE '/u02/oracle/data/tbsa02.dbf' SIZE 1M;
```

You can also change the default storage parameters of a tablespace using the ALTER TABLESPACE statement, as illustrated in the following example:

```
ALTER TABLESPACE users
    DEFAULT STORAGE (
```

```
    NEXT 100K
    MAXEXTENTS 20
    PCTINCREASE 0);
```

New values for the default storage parameters of a tablespace affect only objects that are subsequently created, or extents subsequently allocated for existing segments within the tablespace.

Other reasons for issuing an ALTER TABLESPACE statement include, but are not limited to:

- Coalescing free space in a tablespace. See "Coalescing Free Space in Dictionary-Managed Tablespaces" on page 8-14.
- Altering tablespace availability (ONLINE / OFFLINE). See "Altering Tablespace Availability" on page 8-25.
- Making a tablespace read-only or read / write. See "Using Read-Only Tablespaces" on page 8-27.
- Adding or renaming a datafile, or enabling / disabling the autoextension of the size of a datafile in the tablespace. See Chapter 9, "Managing Datafiles and Tempfiles".

Coalescing Free Space in Dictionary-Managed Tablespaces

Over time, the free space in a dictionary-managed tablespace can become fragmented, making it difficult to allocate new extents. This section discusses how to defragment free space and includes the following topics:

- How Oracle Database Coalesces Free Space
- Manually Coalescing Free Space
- Monitoring Free Space

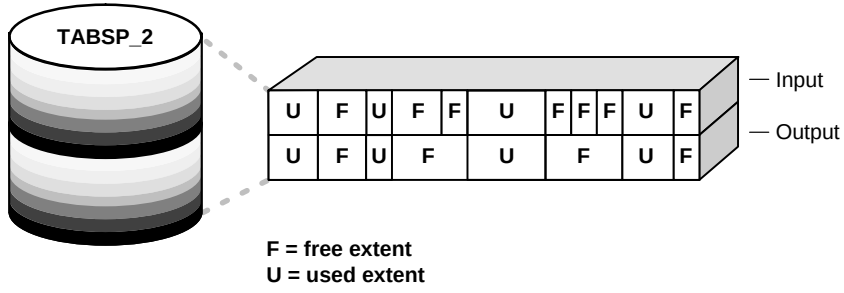
How Oracle Database Coalesces Free Space A free extent in a dictionary-managed tablespace is made up of a collection of contiguous free blocks. When allocating new extents to a tablespace segment, the database uses the free extent closest in size to the required extent. In some cases, when segments are dropped, their extents are deallocated and marked as free, but adjacent free extents are not immediately recombined into larger free extents. The result is fragmentation that makes allocation of larger extents more difficult.

Oracle Database addresses fragmentation in several ways:

- When attempting to allocate a new extent for a segment, the database first tries to find a free extent large enough for the new extent. Whenever the database cannot find a free extent that is large enough for the new extent, it coalesces adjacent free extents in the tablespace and looks again.
- The SMON background process periodically coalesces neighboring free extents when the PCTINCREASE value for a tablespace is not zero. If you set PCTINCREASE=0, no coalescing of free extents occurs. If you are concerned about the overhead of ongoing coalesce operations of SMON, an alternative is to set PCTINCREASE=0, and periodically coalesce free space manually.
- When a segment is dropped or truncated, a limited form of coalescing is performed if the PCTINCREASE value for the segment is not zero. This is done even if PCTINCREASE=0 for the tablespace containing the segment.
- You can use the ALTER TABLESPACE ... COALESCE statement to manually coalesce any adjacent free extents.

The process of coalescing free space is illustrated in the following figure.

Figure 8–1 Coalescing Free Space



Note: Coalescing free space is not necessary for locally managed tablespaces, because bitmaps automatically track adjacent free space.

Manually Coalescing Free Space If you find that fragmentation of space in a tablespace is high (contiguous space on your disk appears as noncontiguous), you can coalesce any free space using the ALTER TABLESPACE ... COALESCE statement. You must have the ALTER TABLESPACE system privilege to coalesce tablespaces.

This statement is useful if PCTINCREASE=0, or you can use it to supplement SMON and extent allocation coalescing. If all extents within the tablespace are of the same size, coalescing is not necessary. This is the case when the default PCTINCREASE value for the tablespace is set to zero, all segments use the default storage parameters of the tablespace, and INITIAL=NEXT=MINIMUM EXTENT.

The following statement coalesces free space in the tablespace tabsp_4:

```
ALTER TABLESPACE tabsp_4 COALESCE;
```

The COALESCE clause of the ALTER TABLESPACE statement is exclusive. You cannot specify any other clause in the same ALTER TABLESPACE statement.

> **Note:** The ALTER TABLESPACE ... COALESCE statement does not coalesce free extents that are separated by data extents. If you observe many free extents located between data extents, you must reorganize the tablespace (for example, by exporting and importing its data) to create useful free space extents.

Monitoring Free Space The following views provide information on the free space in a tablespace:

- DBA_FREE_SPACE
- DBA_FREE_SPACE_COALESCED

The following statement displays the free space in tablespace tabsp_4:

```
SELECT  BLOCK_ID, BYTES, BLOCKS
     FROM  DBA_FREE_SPACE
     WHERE TABLESPACE_NAME = 'TABSP_4'
     ORDER BY BLOCK_ID;

BLOCK_ID   BYTES      BLOCKS
---------- ---------- ----------
         2      16384          2
         4      16384          2
         6      81920         10
        16      16384          2
        27      16384          2
        29      16384          2
        31      16384          2
        33      16384          2
        35      16384          2
        37      16384          2
```

```
        39       8192          1
        40       8192          1
        41     196608         24
13 rows selected.
```

This view shows that there is adjacent free space in `tabsp_4` (for example, blocks starting with BLOCK_IDs 2, 4, 6, 16) that has not been coalesced. After coalescing the tablespace using the ALTER TABLESPACE statement shown previously, the results of this query would read:

```
 BLOCK_ID   BYTES      BLOCKS
---------- ---------- ----------
         2     131072         16
        27     311296         38
2 rows selected.
```

The DBA_FREE_SPACE_COALESCED view displays statistics for coalescing activity. It is also useful in determining if you need to coalesce space.

> **See Also:** *Oracle Database Reference* for more information about these views

Temporary Tablespaces

A **temporary tablespace** contains transient data that persists only for the duration of the session. Temporary tablespaces can improve the concurrence of multiple sort operations, reduce their overhead, and avoid Oracle Database space management operations. A temporary tablespace can be assigned to users with the CREATE USER or ALTER USER statement and can be shared by multiple users.

Within a temporary tablespace, all sort operations for a given instance and tablespace share a single **sort segment**. Sort segments exist for every instance that performs sort operations within a given tablespace. The sort segment is created by the first statement that uses a temporary tablespace for sorting, after startup, and is released only at shutdown. An extent cannot be shared by multiple transactions.

You can view the allocation and deallocation of space in a temporary tablespace sort segment using the V$SORT_SEGMENT view. The V$TEMPSEG_USAGE view identifies the current sort users in those segments.

You cannot explicitly create objects in a temporary tablespace.

See Also:

- *Oracle Database Security Guide* for information about creating users and assigning temporary tablespaces
- *Oracle Database Reference* for more information about the V$SORT_SEGMENT and V$TEMPSEG_USAGE views
- *Oracle Database Performance Tuning Guide* for a discussion on tuning sorts

Creating a Locally Managed Temporary Tablespace

Because space management is much simpler and more efficient in locally managed tablespaces, they are ideally suited for temporary tablespaces. Locally managed temporary tablespaces use **tempfiles**, which do not modify data outside of the temporary tablespace or generate any redo for temporary tablespace data. Because of this, they enable you to perform on-disk sorting operations in a read-only or standby database.

You also use different views for viewing information about tempfiles than you would for datafiles. The V$TEMPFILE and DBA_TEMP_FILES views are analogous to the V$DATAFILE and DBA_DATA_FILES views.

To create a locally managed temporary tablespace, you use the CREATE TEMPORARY TABLESPACE statement, which requires that you have the CREATE TABLESPACE system privilege.

The following statement creates a temporary tablespace in which each extent is 16M. Each 16M extent (which is the equivalent of 8000 blocks when the standard block size is 2K) is represented by a bit in the bitmap for the file.

```
CREATE TEMPORARY TABLESPACE lmtemp TEMPFILE '/u02/oracle/data/lmtemp01.dbf'
     SIZE 20M REUSE
     EXTENT MANAGEMENT LOCAL UNIFORM SIZE 16M;
```

The extent management clause is optional for temporary tablespaces because all temporary tablespaces are created with locally managed extents of a uniform size. The Oracle Database default for SIZE is 1M. But if you want to specify another value for SIZE, you can do so as shown in the preceding statement.

The AUTOALLOCATE clause is not allowed for temporary tablespaces.

> **Note:** On some operating systems, the database does not allocate space for the tempfile until the tempfile blocks are actually accessed. This delay in space allocation results in faster creation and resizing of tempfiles, but it requires that sufficient disk space is available when the tempfiles are later used. Please refer to your operating system documentation to determine whether the database allocates tempfile space in this way on your system.

Creating a Bigfile Temporary Tablespace

Just as for regular tablespaces, you can create single-file (bigfile) temporary tablespaces. Use the CREATE BIGFILE TEMPORARY TABLESPACE statement to create a single-tempfile tablespace. See the sections "Creating a Bigfile Tablespace" on page 8-10 and "Altering a Bigfile Tablespace" on page 8-11 for information about bigfile tablespaces, but consider that you are creating temporary tablespaces that use tempfiles instead of datafiles.

Altering a Locally Managed Temporary Tablespace

Except for adding a tempfile, as illustrated in the following example, you cannot use the ALTER TABLESPACE statement for a locally managed temporary tablespace.

```
ALTER TABLESPACE lmtemp
   ADD TEMPFILE '/u02/oracle/data/lmtemp02.dbf' SIZE 18M REUSE;
```

> **Note:** You cannot use the ALTER TABLESPACE statement, with the TEMPORARY keyword, to change a locally managed permanent tablespace into a locally managed temporary tablespace. You must use the CREATE TEMPORARY TABLESPACE statement to create a locally managed temporary tablespace.

However, the ALTER DATABASE statement can be used to alter tempfiles.

The following statements take offline and bring online temporary files:

```
ALTER DATABASE TEMPFILE '/u02/oracle/data/lmtemp02.dbf' OFFLINE;
ALTER DATABASE TEMPFILE '/u02/oracle/data/lmtemp02.dbf' ONLINE;
```

The following statement resizes a temporary file:

```
ALTER DATABASE TEMPFILE '/u02/oracle/data/lmtemp02.dbf' RESIZE 18M;
```

The following statement drops a temporary file and deletes the operating system file:

```
ALTER DATABASE TEMPFILE '/u02/oracle/data/lmtemp02.dbf' DROP
    INCLUDING DATAFILES;
```

The tablespace to which this tempfile belonged remains. A message is written to the alert file for the datafile that was deleted. If an operating system error prevents the deletion of the file, the statement still succeeds, but a message describing the error is written to the alert file.

It is also possible, but not shown, to use the ALTER DATABASE statement to enable or disable the automatic extension of an existing tempfile, and to rename (RENAME FILE) a tempfile.

Creating a Dictionary-Managed Temporary Tablespace

In earlier releases, you could create a dictionary-managed temporary tablespace by specifying the TEMPORARY keyword after specifying the tablespace name in a CREATE TABLESPACE statement. This syntax has been deprecated. It is still supported in case you are using dictionary-managed tablespaces, which are not supported by the CREATE TEMPORARY TABLEPSPACE syntax. If you do use this deprecated syntax, the extent cannot be locally managed, nor can you specify a nonstandard block size for the tablespace.

Oracle strongly recommends that you create locally managed temporary tablespaces containing tempfiles, as described in the preceding sections. The creation of new dictionary-managed tablespaces is scheduled for desupport.

> **See Also:** *Oracle Database SQL Reference* for syntax and semantics of the CREATE TEMPORARY TABLESPACE and CREATE TABLESPACE . . . TEMPORARY statements

Altering a Dictionary-Managed Temporary Tablespace

You can issue the ALTER TABLESPACE statement against a dictionary-managed temporary tablespace using many of the same keywords and clauses as for a permanent dictionary-managed tablespace. Restrictions are noted in the *Oracle Database SQL Reference*.

> **Note:** When you take dictionary-managed temporary tablespaces offline with the ALTER TABLESPACE ... OFFLINE statement, returning them online does not affect their temporary status.

You can change an existing permanent dictionary-managed tablespace to a temporary tablespace, using the ALTER TABLESPACE statement. For example:

```
ALTER TABLESPACE tbsa TEMPORARY;
```

Multiple Temporary Tablespaces: Using Tablespace Groups

A **tablespace group** enables a user to consume temporary space from multiple tablespaces. A tablespace group has the following characteristics:

- It contains at least one tablespace. There is no explicit limit on the maximum number of tablespaces that are contained in a group.
- It shares the namespace of tablespaces, so its name cannot be the same as any tablespace.
- You can specify a tablespace group name wherever a tablespace name would appear when you assign a default temporary tablespace for the database or a temporary tablespace for a user.

You do not explicitly create a tablespace group. Rather, it is created implicitly when you assign the first temporary tablespace to the group. The group is deleted when the last temporary tablespace it contains is removed from it.

Using a tablespace group, rather than a single temporary tablespace, can alleviate problems caused where one tablespace is inadequate to hold the results of a sort, particularly on a table that has many partitions. A tablespace group enables parallel execution servers in a single parallel operation to use multiple temporary tablespaces.

The view DBA_TABLESPACE_GROUPS lists tablespace groups and their member tablespaces.

> **See Also:** *Oracle Database Security Guide* for more information about assigning a temporary tablespace or tablespace group to a user

Creating a Tablespace Group

You create a tablespace group implicitly when you include the TABLESPACE GROUP clause in the CREATE TEMPORARY TABLESPACE or ALTER TABLESPACE statement and the specified tablespace group does not currently exist.

For example, if neither group1 nor group2 exists, then the following statements create those groups, each of which has only the specified tablespace as a member:

```
CREATE TEMPORARY TABLESPACE lmtemp2 TEMPFILE '/u02/oracle/data/lmtemp201.dbf'
     SIZE 50M
     TABLESPACE GROUP group1;

ALTER TABLESPACE lmtemp TABLESPACE GROUP group2;
```

Changing Members of a Tablespace Group

You can add a tablespace to an existing tablespace group by specifying the existing group name in the TABLESPACE GROUP clause of the CREATE TEMPORARY TABLESPACE or ALTER TABLESPACE statement.

The following statement adds a tablespace to an existing group. It creates and adds tablespace lmtemp3 to group1, so that group1 contains tablespaces lmtemp2 and lmtemp3.

```
CREATE TEMPORARY TABLESPACE lmtemp3 TEMPFILE '/u02/oracle/data/lmtemp301.dbf'
     SIZE 25M
     TABLESPACE GROUP group1;
```

The following statement also adds a tablespace to an existing group, but in this case because tablespace lmtemp2 already belongs to group1, it is in effect moved from group1 to group2:

```
ALTER TABLESPACE lmtemp2 TABLESPACE GROUP group2;
```

Now group2 contains both lmtemp and lmtemp2, while group1 consists of only tmtemp3.

You can remove a tablespace from a group as shown in the following statement:

```
ALTER TABLESPACE lmtemp3 TABLESPACE GROUP '';
```

Tablespace lmtemp3 no longer belongs to any group. Further, since there are no longer any members of group1, this results in the implicit deletion of group1.

Assigning a Tablespace Group as the Default Temporary Tablespace

Use the ALTER DATABASE ...DEFAULT TEMPORARY TABLESPACE statement to assign a tablespace group as the default temporary tablespace for the database. For example:

```
ALTER DATABASE sample DEFAULT TEMPORARY TABLESPACE group2;
```

Any user who has not explicitly been assigned a temporary tablespace will now use tablespaces lmtemp and lmtemp2.

If a tablespace group is specified as the default temporary tablespace, you cannot drop any of its member tablespaces. You must first be remove from the tablespace from the tablespace group. Likewise, you cannot drop a single temporary as long as it is the default temporary tablespace.

Specifying Nonstandard Block Sizes for Tablespaces

You can create tablespaces with block sizes different from the standard database block size, which is specified by the DB_BLOCK_SIZE initialization parameter. This feature lets you transport tablespaces with unlike block sizes between databases.

Use the BLOCKSIZE clause of the CREATE TABLESPACE statement to create a tablespace with a block size different from the database standard block size. In order for the BLOCKSIZE clause to succeed, you must have already set the DB_CACHE_SIZE and at least one DB_nK_CACHE_SIZE initialization parameter. Further, and the integer you specify in the BLOCKSIZE clause must correspond with the setting of one DB_nK_CACHE_SIZE parameter setting. Although redundant, specifying a BLOCKSIZE equal to the standard block size, as specified by the DB_BLOCK_SIZE initialization parameter, is allowed.

The following statement creates tablespace lmtbsb, but specifies a block size that differs from the standard database block size (as specified by the DB_BLOCK_SIZE initialization parameter):

```
CREATE TABLESPACE lmtbsb DATAFILE '/u02/oracle/data/lmtbsb01.dbf' SIZE 50M
    EXTENT MANAGEMENT LOCAL UNIFORM SIZE 128K
    BLOCKSIZE 8K;
```

See Also:

- "Specifying Database Block Sizes" on page 2-31
- "Setting the Buffer Cache Initialization Parameters" on page 2-39 for information about the DB_CACHE_SIZE and DB_nK_CACHE_SIZE parameter settings
- "Transporting Tablespaces Between Databases" on page 8-40

Controlling the Writing of Redo Records

For some database operations, you can control whether the database generates redo records. Without redo, no media recovery is possible. However, suppressing redo generation can improve performance, and may be appropriate for easily recoverable operations. An example of such an operation is a CREATE TABLE...AS SELECT statement, which can be repeated in case of database or instance failure.

Specify the NOLOGGING clause in the CREATE TABLESPACE statement if you wish to suppress redo when these operations are performed for objects within the tablespace. If you do not include this clause, or if you specify LOGGING instead, then the database generates redo when changes are made to objects in the tablespace. Redo is never generated for temporary segments or in temporary tablespaces, regardless of the logging attribute.

The logging attribute specified at the tablespace level is the default attribute for objects created within the tablespace. You can override this default logging attribute by specifying LOGGING or NOLOGGING at the schema object level--for example, in a CREATE TABLE statement.

If you have a standby database, NOLOGGING mode causes problems with the availability and accuracy of the standby database. To overcome this problem, you can specify FORCE LOGGING mode. When you include the FORCE LOGGING clause in the CREATE TABLESPACE statement, you force the generation of redo records for all operations that make changes to objects in a tablespace. This overrides any specification made at the object level.

If you transport a tablespace that is in FORCE LOGGING mode to another database, the new tablespace will not maintain the FORCE LOGGING mode.

See Also:

- *Oracle Database SQL Reference* for information about operations that can be done in `NOLOGGING` mode
- "Specifying FORCE LOGGING Mode" on page 2-26 for more information about `FORCE LOGGING` mode and for information about the effects of the `FORCE LOGGING` clause used with the `CREATE DATABASE` statement

Altering Tablespace Availability

You can take an online tablespace offline so that it is temporarily unavailable for general use. The rest of the database remains open and available for users to access data. Conversely, you can bring an offline tablespace online to make the schema objects within the tablespace available to database users. The database must be open to alter the availability of a tablespace.

To alter the availability of a tablespace, use the `ALTER TABLESPACE` statement. You must have the `ALTER TABLESPACE` or `MANAGE TABLESPACE` system privilege.

See Also: "Altering Datafile Availability" on page 9-8 for information about altering the availability of individual datafiles within a tablespace

Taking Tablespaces Offline

You may want to take a tablespace offline for any of the following reasons:

- To make a portion of the database unavailable while allowing normal access to the remainder of the database
- To perform an offline tablespace backup (even though a tablespace can be backed up while online and in use)
- To make an application and its group of tables temporarily unavailable while updating or maintaining the application

When a tablespace is taken offline, the database takes all the associated files offline.

You cannot take the following tablespaces offline:

- `SYSTEM`
- The undo tablespace
- Temporary tablespaces

Before taking a tablespace offline, consider altering the tablespace allocation of any users who have been assigned the tablespace as a default tablespace. Doing so is advisable because those users will not be able to access objects in the tablespace while it is offline.

You can specify any of the following parameters as part of the ALTER TABLESPACE ... OFFLINE statement:

Clause	Description
NORMAL	A tablespace can be taken offline normally if no error conditions exist for any of the datafiles of the tablespace. No datafile in the tablespace can be currently offline as the result of a write error. When you specify OFFLINE NORMAL, the database takes a checkpoint for all datafiles of the tablespace as it takes them offline. NORMAL is the default.
TEMPORARY	A tablespace can be taken offline temporarily, even if there are error conditions for one or more files of the tablespace. When you specify OFFLINE TEMPORARY, the database takes offline the datafiles that are not already offline, checkpointing them as it does so. If no files are offline, but you use the temporary clause, media recovery is not required to bring the tablespace back online. However, if one or more files of the tablespace are offline because of write errors, and you take the tablespace offline temporarily, the tablespace requires recovery before you can bring it back online.
IMMEDIATE	A tablespace can be taken offline immediately, without the database taking a checkpoint on any of the datafiles. When you specify OFFLINE IMMEDIATE, media recovery for the tablespace is required before the tablespace can be brought online. You cannot take a tablespace offline immediately if the database is running in NOARCHIVELOG mode.

Caution: If you must take a tablespace offline, use the NORMAL clause (the default) if possible. This setting guarantees that the tablespace will not require recovery to come back online, even if after incomplete recovery you reset the redo log sequence using an ALTER DATABASE OPEN RESETLOGS statement.

Specify TEMPORARY only when you cannot take the tablespace offline normally. In this case, only the files taken offline because of errors need to be recovered before

the tablespace can be brought online. Specify `IMMEDIATE` only after trying both the normal and temporary settings.

The following example takes the `users` tablespace offline normally:

```
ALTER TABLESPACE users OFFLINE NORMAL;
```

Bringing Tablespaces Online

You can bring any tablespace in an Oracle Database online whenever the database is open. A tablespace is normally online so that the data contained within it is available to database users.

If a tablespace to be brought online was not taken offline "cleanly" (that is, using the `NORMAL` clause of the `ALTER TABLESPACE OFFLINE` statement), you must first perform media recovery on the tablespace before bringing it online. Otherwise, the database returns an error and the tablespace remains offline.

> **See Also:** Depending upon your archiving strategy, refer to one of the following books for information about performing media recovery:
>
> - *Oracle Database Backup and Recovery Basics*
> - *Oracle Database Backup and Recovery Advanced User's Guide*

The following statement brings the `users` tablespace online:

```
ALTER TABLESPACE users ONLINE;
```

Using Read-Only Tablespaces

Making a tablespace read-only prevents write operations on the datafiles in the tablespace. The primary purpose of read-only tablespaces is to eliminate the need to perform backup and recovery of large, static portions of a database. Read-only tablespaces also provide a way to protecting historical data so that users cannot modify it. Making a tablespace read-only prevents updates on all tables in the tablespace, regardless of a user's update privilege level.

Note: Making a tablespace read-only cannot in itself be used to satisfy archiving or data publishing requirements, because the tablespace can only be brought online in the database in which it was created. However, you can meet such requirements by using the transportable tablespace feature, as described in "Transporting Tablespaces Between Databases" on page 8-40.

You can drop items, such as tables or indexes, from a read-only tablespace, but you cannot create or alter objects in a read-only tablespace. You can execute statements that update the file description in the data dictionary, such as `ALTER TABLE ... ADD` or `ALTER TABLE ... MODIFY`, but you will not be able to utilize the new description until the tablespace is made read/write.

Read-only tablespaces can be transported to other databases. And, since read-only tablespaces can never be updated, they can reside on CD-ROM or WORM (Write Once-Read Many) devices.

The following topics are discussed in this section:

- Making a Tablespace Read-Only
- Making a Read-Only Tablespace Writable
- Creating a Read-Only Tablespace on a WORM Device
- Delaying the Opening of Datafiles in Read-Only Tablespaces

See Also: "Transporting Tablespaces Between Databases" on page 8-40

Making a Tablespace Read-Only

All tablespaces are initially created as read/write. Use the `READ ONLY` clause in the `ALTER TABLESPACE` statement to change a tablespace to read-only. You must have the `ALTER TABLESPACE` or `MANAGE TABLESPACE` system privilege.

Before you can make a tablespace read-only, the following conditions must be met.

- The tablespace must be online. This is necessary to ensure that there is no undo information that needs to be applied to the tablespace.
- The tablespace cannot be the active undo tablespace or `SYSTEM` tablespace.
- The tablespace must not currently be involved in an online backup, because the end of a backup updates the header file of all datafiles in the tablespace.

For better performance while accessing data in a read-only tablespace, you can issue a query that accesses all of the blocks of the tables in the tablespace just before making it read-only. A simple query, such as SELECT COUNT (*), executed against each table ensures that the data blocks in the tablespace can be subsequently accessed most efficiently. This eliminates the need for the database to check the status of the transactions that most recently modified the blocks.

The following statement makes the flights tablespace read-only:

```
ALTER TABLESPACE flights READ ONLY;
```

You can issue the ALTER TABLESPACE ... READ ONLY statement while the database is processing transactions. Once the statement is issued, no transactions are allowed to make further changes (using DML statements) to the tablespace being made read-only. Transactions that have already made changes to the tablespace are allowed to commit or terminate.

If any transactions are ongoing, then the ALTER TABLESPACE ... READ ONLY statement may not return instantaneously. The database waits for all transactions started before you issued READ ONLY to either commit or terminate.

Note: This transitional read-only state only occurs if the value of the initialization parameter COMPATIBLE is 8.1.0 or greater. If this parameter is set to a value less than 8.1.0, the ALTER TABLESPACE ... READ ONLY statement fails if any active transactions exist.

If you find it is taking a long time for the ALTER TABLESPACE statement to complete, you can identify the transactions that are preventing the read-only state from taking effect. You can then notify the owners of those transactions and decide whether to terminate the transactions, if necessary.

The following example identifies the transaction entry for the ALTER TABLESPACE ... READ ONLY statement and note its session address (saddr):

```
SELECT SQL_TEXT, SADDR
    FROM V$SQLAREA,V$SESSION
    WHERE V$SQLAREA.ADDRESS = V$SESSION.SQL_ADDRESS
        AND SQL_TEXT LIKE 'alter tablespace%';

SQL_TEXT                                 SADDR
---------------------------------------- --------
alter tablespace tbs1 read only          80034AF0
```

The start SCN of each active transaction is stored in the V$TRANSACTION view. Displaying this view sorted by ascending start SCN lists the transactions in execution order. From the preceding example, you already know the session address of the transaction entry for the read-only statement, and you can now locate it in the V$TRANSACTION view. All transactions with smaller start SCN, which indicates an earlier execution, can potentially hold up the quiesce and subsequent read-only state of the tablespace.

```
SELECT SES_ADDR, START_SCNB
    FROM V$TRANSACTION
    ORDER BY START_SCNB;

SES_ADDR START_SCNB
-------- ----------
800352A0       3621   --> waiting on this txn
80035A50       3623   --> waiting on this txn
80034AF0       3628   --> this is the ALTER TABLESPACE statement
80037910       3629   --> don't care about this txn
```

After making the tablespace read-only, it is advisable to back it up immediately. As long as the tablespace remains read-only, no further backups of the tablespace are necessary, because no changes can be made to it.

> **See Also:** Depending upon your backup and recovery strategy, refer to one of the following books for information about backing up and recovering a database with read-only datafiles:
>
> - *Oracle Database Backup and Recovery Basics*
> - *Oracle Database Backup and Recovery Advanced User's Guide*

Making a Read-Only Tablespace Writable

Use the READ WRITE keywords in the ALTER TABLESPACE statement to change a tablespace to allow write operations. You must have the ALTER TABLESPACE or MANAGE TABLESPACE system privilege.

A prerequisite to making the tablespace read / write is that all of the datafiles in the tablespace, as well as the tablespace itself, must be online. Use the DATAFILE ... ONLINE clause of the ALTER DATABASE statement to bring a datafile online. The V$DATAFILE view lists the current status of datafiles.

The following statement makes the `flights` tablespace writable:

```
ALTER TABLESPACE flights READ WRITE;
```

Making a read-only tablespace writable updates the control file entry for the datafiles, so that you can use the read-only version of the datafiles as a starting point for recovery.

Creating a Read-Only Tablespace on a WORM Device

Follow these steps to create a read-only tablespace on a CD-ROM or WORM (Write Once-Read Many) device.

1. Create a writable tablespace on another device. Create the objects that belong in the tablespace and insert your data.
2. Alter the tablespace to make it read-only.
3. Copy the datafiles of the tablespace onto the WORM device. Use operating system commands to copy the files.
4. Take the tablespace offline.
5. Rename the datafiles to coincide with the names of the datafiles you copied onto your WORM device. Use `ALTER TABLESPACE` with the `RENAME DATAFILE` clause. Renaming the datafiles changes their names in the control file.
6. Bring the tablespace back online.

Delaying the Opening of Datafiles in Read-Only Tablespaces

When substantial portions of a very large database are stored in read-only tablespaces that are located on slow-access devices or hierarchical storage, you should consider setting the `READ_ONLY_OPEN_DELAYED` initialization parameter to `TRUE`. This speeds certain operations, primarily opening the database, by causing datafiles in read-only tablespaces to be accessed for the first time only when an attempt is made to read data stored within them.

Setting `READ_ONLY_OPEN_DELAYED=TRUE` has the following side-effects:

- A missing or bad read-only file is not detected at open time. It is only discovered when there is an attempt to access it.
- `ALTER SYSTEM CHECK DATAFILES` does not check read-only files.
- `ALTER TABLESPACE ... ONLINE` and `ALTER DATABASE DATAFILE ... ONLINE` do not check read-only files. They are checked only upon the first access.

- V$RECOVER_FILE, V$BACKUP, and V$DATAFILE_HEADER do not access read-only files. Read-only files are indicated in the results list with the error "DELAYED OPEN", with zeroes for the values of other columns.
- V$DATAFILE does not access read-only files. Read-only files have a size of "0" listed.
- V$RECOVER_LOG does not access read-only files. Logs they could need for recovery are not added to the list.
- ALTER DATABASE NOARCHIVELOG does not access read-only files.It proceeds even if there is a read-only file that requires recovery.

Notes:

- RECOVER DATABASE and ALTER DATABASE OPEN RESETLOGS continue to access all read-only datafiles regardless of the parameter value. If you want to avoid accessing read-only files for these operations, those files should be taken offline.
- If a backup control file is used, the read-only status of some files may be inaccurate. This can cause some of these operations to return unexpected results. Care should be taken in this situation.

Renaming Tablespaces

Using the RENAME TO clause of the ALTER TABLESPACE, you can rename a permanent or temporary tablespace. For example, the following statement renames the users tablespace:

```
ALTER TABLESPACE users RENAME TO usersts;
```

When you rename a tablespace the database updates all references to the tablespace name in the data dictionary, control file, and (online) datafile headers. The database does not change the tablespace ID so if this tablespace were, for example, the default tablespace for a user, then the renamed tablespace would show as the default tablespace for the user in the DBA_USERS view.

The following affect the operation of this statement:

- The COMPATIBLE parameter must be set to 10.0 or higher.

- If the tablespace being renamed is the SYSTEM tablespace or the SYSAUX tablespace, then it will not be renamed and an error is raised.
- If any datafile in the tablespace is offline, or if the tablespace is offline, then the tablespace is not renamed and an error is raised.
- If the tablespace is read only, then datafile headers are not updated. This should not be regarded as corruption; instead, it causes a message to be written to the alert log indicating that datafile headers have not been renamed. The data dictionary and control file are updated.
- If the tablespace is the default temporary tablespace, then the corresponding entry in the database properties table is updated and the DATABASE_PROPERTIES view shows the new name.
- If the tablespace is an undo tablespace and if the following conditions are met, then the tablespace name is changed to the new tablespace name in the server parameter file (SPFILE).
  - The server parameter file was used to start up the database.
  - The tablespace name is specified as the UNDO_TABLESPACE for any instance.

  If a traditional initialization parameter file (PFILE) is being used then a message is written to the alert file stating that the initialization parameter file must be manually changed.

Dropping Tablespaces

You can drop a tablespace and its contents (the segments contained in the tablespace) from the database if the tablespace and its contents are no longer required. You must have the DROP TABLESPACE system privilege to drop a tablespace.

Caution: Once a tablespace has been dropped, the data in the tablespace is not recoverable. Therefore, make sure that all data contained in a tablespace to be dropped will not be required in the future. Also, immediately before and after dropping a tablespace from a database, back up the database completely. This is *strongly recommended* so that you can recover the database if you mistakenly drop a tablespace, or if the database experiences a problem in the future after the tablespace has been dropped.

When you drop a tablespace, the file pointers in the control file of the associated database are removed. You can optionally direct Oracle Database to delete the operating system files (datafiles) that constituted the dropped tablespace. If you do not direct the database to delete the datafiles at the same time that it deletes the tablespace, you must later use the appropriate commands of your operating system to delete them.

You cannot drop a tablespace that contains any active segments. For example, if a table in the tablespace is currently being used or the tablespace contains undo data needed to roll back uncommitted transactions, you cannot drop the tablespace. The tablespace can be online or offline, but it is best to take the tablespace offline before dropping it.

To drop a tablespace, use the DROP TABLESPACE statement. The following statement drops the users tablespace, including the segments in the tablespace:

```
DROP TABLESPACE users INCLUDING CONTENTS;
```

If the tablespace is empty (does not contain any tables, views, or other structures), you do not need to specify the INCLUDING CONTENTS clause. Use the CASCADE CONSTRAINTS clause to drop all referential integrity constraints from tables outside the tablespace that refer to primary and unique keys of tables inside the tablespace.

To delete the datafiles associated with a tablespace at the same time that the tablespace is dropped, use the INCLUDING CONTENTS AND DATAFILES clause. The following statement drops the users tablespace and its associated datafiles:

```
DROP TABLESPACE users INCLUDING CONTENTS AND DATAFILES;
```

A message is written to the alert file for each datafile that is deleted. If an operating system error prevents the deletion of a file, the DROP TABLESPACE statement still succeeds, but a message describing the error is written to the alert file.

Managing the SYSAUX Tablespace

The SYSAUX tablespace was installed as an auxiliary tablespace to the SYSTEM tablespace when you created your database. Some database components that formerly created and used separate tablespaces now occupy the SYSAUX tablespace.

If the SYSAUX tablespace becomes unavailable, core database functionality will remain operational. The database features that use the SYSAUX tablespace could fail, or function with limited capability.

Monitoring Occupants of the SYSAUX Tablespace

The list of registered occupants of the SYSAUX tablespace are discussed in "Creating the SYSAUX Tablespace" on page 2-17. These components can use the SYSAUX tablespace, and their installation provides the means of establishing their occupancy of the SYSAUX tablespace.

You can monitor the occupants of the SYSAUX tablespace using the V$SYSAUX_OCCUPANTS view. This view lists the following information about the occupants of the SYSAUX tablespace:

- Name of the occupant
- Occupant description
- Schema name
- Move procedure
- Current space usage

View information is maintained by the occupants.

> **See Also:** *Oracle Database Reference* for a detailed description of the V$SYSAUX_OCCUPANTS view

Moving Occupants Out Of or Into the SYSAUX Tablespace

You will have an option at component install time to specify that you do not want the component to reside in SYSAUX. Also, if you later decide that the component should be relocated to a designated tablespace, you can use the move procedure for that component, as specified in the V$SYSAUX_OCCUPANTS view, to perform the move.

For example, assume that you install Oracle Ultra Search into the default tablespace, which is SYSAUX. Later you discover that Ultra Search is using up too much space. To alleviate this space pressure on SYSAUX, you can call a PL/SQL move procedure specified in the V$SYSAUX_OCCUPANTS view to relocate Ultra Search to another tablespace.

The move procedure also lets you move a component from another tablespace into the SYSAUX tablespace.

Controlling the Size of the SYSAUX Tablespace

The SYSAUX tablespace is occupied by a number of database components (see Table 2–2), and its total size is governed by the space consumed by those components. The space consumed by the components, in turn, depends on which features or functionality are being used and on the nature of the database workload.

The largest portion of the SYSAUX tablespace is occupied by the Automatic Workload Repository (AWR). The space consumed by the AWR is determined by several factors, including the number of active sessions in the system at any given time, the snapshot interval, and the historical data retention period. A typical system with an average of 30 concurrent active sessions may require approximately 200 to 300 MB of space for its AWR data. You can control the size of the AWR by changing the snapshot interval and historical data retention period. For more information on managing the AWR snapshot interval and retention period, please refer to *Oracle Database Performance Tuning Guide*.

Another major occupant of the SYSAUX tablespace is the embedded Enterprise Manager (EM) repository. This repository is used by Oracle Enterprise Manager Database Control to store its metadata. The size of this repository depends on database activity and on configuration-related information stored in the repository.

Other database components in the SYSAUX tablespace will grow in size only if their associated features (for example, Oracle UltraSearch, Oracle Text, Oracle Streams) are in use. If the features are not used, then these components do not have any significant effect on the size of the SYSAUX tablespace.

Diagnosing and Repairing Locally Managed Tablespace Problems

Note: The DBMS_SPACE_ADMIN package provides administrators with defect diagnosis and repair functionality for locally managed tablespaces. It cannot be used in this capacity for dictionary-managed tablespaces.

It also provides procedures for migrating from dictionary-managed tablespaces to locally managed tablespaces, and the reverse.

The DBMS_SPACE_ADMIN package contains the following procedures:

Procedure	Description
SEGMENT_VERIFY	Verifies the consistency of the extent map of the segment.
SEGMENT_CORRUPT	Marks the segment corrupt or valid so that appropriate error recovery can be done. Cannot be used for a locally managed SYSTEM tablespace.
SEGMENT_DROP_CORRUPT	Drops a segment currently marked corrupt (without reclaiming space). Cannot be used for a locally managed SYSTEM tablespace.
SEGMENT_DUMP	Dumps the segment header and extent map of a given segment.
TABLESPACE_VERIFY	Verifies that the bitmaps and extent maps for the segments in the tablespace are in sync.
TABLESPACE_REBUILD_BITMAPS	Rebuilds the appropriate bitmap. Cannot be used for a locally managed SYSTEM tablespace.
TABLESPACE_FIX_BITMAPS	Marks the appropriate data block address range (extent) as free or used in bitmap. Cannot be used for a locally managed SYSTEM tablespace.
TABLESPACE_REBUILD_QUOTAS	Rebuilds quotas for given tablespace.
TABLESPACE_MIGRATE_FROM_LOCAL	Migrates a locally managed tablespace to dictionary-managed tablespace. Cannot be used to migrate a locally managed SYSTEM tablespace to a dictionary-managed SYSTEM tablespace.
TABLESPACE_MIGRATE_TO_LOCAL	Migrates a tablespace from dictionary-managed format to locally managed format.
TABLESPACE_RELOCATE_BITMAPS	Relocates the bitmaps to the destination specified. Cannot be used for a locally managed system tablespace.
TABLESPACE_FIX_SEGMENT_STATES	Fixes the state of the segments in a tablespace in which migration was aborted.

The following scenarios describe typical situations in which you can use the DBMS_SPACE_ADMIN package to diagnose and resolve problems.

Note: Some of these procedures can result in lost and unrecoverable data if not used properly. You should work with Oracle Support Services if you have doubts about these procedures.

See Also: *PL/SQL Packages and Types Reference* for details about the DBMS_SPACE_ADMIN package

Scenario 1: Fixing Bitmap When Allocated Blocks are Marked Free (No Overlap)

The TABLESPACE_VERIFY procedure discovers that a segment has allocated blocks that are marked free in the bitmap, but no overlap between segments is reported.

In this scenario, perform the following tasks:

1. Call the SEGMENT_DUMP procedure to dump the ranges that the administrator allocated to the segment.
2. For each range, call the TABLESPACE_FIX_BITMAPS procedure with the TABLESPACE_EXTENT_MAKE_USED option to mark the space as used.
3. Call TABLESPACE_REBUILD_QUOTAS to fix up quotas.

Scenario 2: Dropping a Corrupted Segment

You cannot drop a segment because the bitmap has segment blocks marked "free". The system has automatically marked the segment corrupted.

In this scenario, perform the following tasks:

1. Call the SEGMENT_VERIFY procedure with the SEGMENT_VERIFY_EXTENTS_GLOBAL option. If no overlaps are reported, then proceed with steps 2 through 5.
2. Call the SEGMENT_DUMP procedure to dump the DBA ranges allocated to the segment.
3. For each range, call TABLESPACE_FIX_BITMAPS with the TABLESPACE_EXTENT_MAKE_FREE option to mark the space as free.
4. Call SEGMENT_DROP_CORRUPT to drop the SEG$ entry.
5. Call TABLESPACE_REBUILD_QUOTAS to fix up quotas.

Scenario 3: Fixing Bitmap Where Overlap is Reported

The TABLESPACE_VERIFY procedure reports some overlapping. Some of the real data must be sacrificed based on previous internal errors.

After choosing the object to be sacrificed, in this case say, table t1, perform the following tasks:

1. Make a list of all objects that t1 overlaps.
2. Drop table t1. If necessary, follow up by calling the SEGMENT_DROP_CORRUPT procedure.
3. Call the SEGMENT_VERIFY procedure on all objects that t1 overlapped. If necessary, call the TABLESPACE_FIX_BITMAPS procedure to mark appropriate bitmap blocks as used.
4. Rerun the TABLESPACE_VERIFY procedure to verify the problem is resolved.

Scenario 4: Correcting Media Corruption of Bitmap Blocks

A set of bitmap blocks has media corruption.

In this scenario, perform the following tasks:

1. Call the TABLESPACE_REBUILD_BITMAPS procedure, either on all bitmap blocks, or on a single block if only one is corrupt.
2. Call the TABLESPACE_REBUILD_QUOTAS procedure to rebuild quotas.
3. Call the TABLESPACE_VERIFY procedure to verify that the bitmaps are consistent.

Scenario 5: Migrating from a Dictionary-Managed to a Locally Managed Tablespace

You migrate a dictionary-managed tablespace to a locally managed tablespace. You use the TABLESPACE_MIGRATE_TO_LOCAL procedure. This operation is done online, but space management operations are blocked pending completion of the migration. In other words, you can read or modify data while the migration is in progress, but if you are loading a large amount of data that requires the allocation of additional extents, then the operation may be blocked.

Let us assume that the database block size is 2K, and the existing extent sizes in tablespace tbs_1 are 10, 50, and 10,000 blocks (used, used, and free). The MINIMUM EXTENT value is 20K (10 blocks). In this scenario, you allow the bitmap allocation unit to be chosen by the system. The value of 10 blocks is chosen, because it is the highest common denominator and does not exceed MINIMUM EXTENT.

The statement to convert tbs_1 to a locally managed tablespace is as follows:

```
EXEC DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_TO_LOCAL ('tbs_1');
```

If you choose to specify an allocation unit size, it must be a factor of the unit size calculated by the system, otherwise an error message is issued.

Migrating the SYSTEM Tablespace to a Locally Managed Tablespace

Use the DBMS_SPACE_ADMIN package to migrate the SYSTEM tablespace from dictionary-managed to locally managed. The following statement performs the migration:

```
SQL> EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_TO_LOCAL('SYSTEM');
```

Before performing the migration the following conditions must be met:

- The database has a default temporary tablespace that is not SYSTEM.
- There are no rollback segments in dictionary-managed tablespaces.
- There is at least one online rollback segment in a locally managed tablespace, or if using automatic undo management, an undo tablespace is online.
- All tablespaces other than the tablespace containing the undo space (that is, the tablespace containing the rollback segment or the undo tablespace) are in read-only mode.
- There is a cold backup of the database.
- The system is in restricted mode.

All of these conditions, except for the cold backup, are enforced by the TABLESPACE_MIGRATE_TO_LOCAL procedure.

Note: After the SYSTEM tablespace is migrated to locally managed, any dictionary-managed tablespaces in the database cannot be made read/write. If you want to be able to use the dictionary-managed tablespaces in read/write mode, Oracle recommends that you first migrate these tablespaces to locally managed before migrating the SYSTEM tablespace.

Transporting Tablespaces Between Databases

This section describes how to transport tablespaces between databases, and contains the following topics:

Introduction to Transportable Tablespaces

> **Note:** You must be using the Enterprise Edition of Oracle8*i* or higher to generate a transportable tablespace set. However, you can use any edition of Oracle8*i* or higher to plug a transportable tablespace set into an Oracle Database on the same platform. To plug a transportable tablespace set into an Oracle Database on a different platform, both databases must have compatibility set to at least 10.0. Please refer to "Compatibility Considerations for Transportable Tablespaces" on page 8-44 for a discussion of database compatibility for transporting tablespaces across release levels.

You can use the transportable tablespaces feature to move a subset of an Oracle Database and "plug" it in to another Oracle Database, essentially moving tablespaces between the databases. The tablespaces being transported can be either dictionary managed or locally managed. Starting with Oracle9*i*, the transported tablespaces are not required to be of the same block size as the target database standard block size.

Moving data using transportable tablespaces is much faster than performing either an export/import or unload/load of the same data. This is because the datafiles containing all of the actual data are simply copied to the destination location, and you use an import utility to transfer only the metadata of the tablespace objects to the new database.

> **Note:** The remainder of this chapter uses Data Pump as the import/export utility to use. However, the transportable tablespaces feature supports both Data Pump and the original import and export utilities. Please refer to *Oracle Database Utilities* for more information on these utilities.

The transportable tablespace feature is useful in a number of scenarios, including:

- Exporting and importing partitions in data warehousing tables

- Publishing structured data on CDs
- Copying multiple read-only versions of a tablespace on multiple databases
- Archiving historical data
- Performing tablespace point-in-time-recovery (TSPITR)

These scenarios are discussed in "Using Transportable Tablespaces: Scenarios" on page 8-54

> **See Also:** *Oracle Data Warehousing Guide* for information about using transportable tablespaces in a data warehousing environment

About Transporting Tablespaces Across Platforms

Starting with Oracle Database 10*g*, you can transport tablespaces across platforms. This functionality can be used to:

- Allow a database to be migrated from one platform to another
- Provide an easier and more efficient means for content providers to publish structured data and distribute it to customers running Oracle Database on different platforms
- Simplify the distribution of data from a data warehouse environment to data marts, which are often running on smaller platforms
- Enable the sharing of read-only tablespaces between Oracle Database installations on different operating systems or platforms, assuming that your storage system is accessible from those platforms and the platforms all have the same endianness, as described in the sections that follow

Many, but not all, platforms are supported for cross-platform tablespace transport. You can query the V$TRANSPORTABLE_PLATFORM view to see the platforms that are supported, and to determine their platform IDs and their endian format (byte ordering). For example, the following query displays the currently supported platforms:

```
SQL> COLUMN PLATFORM_NAME FORMAT A30
SQL> SELECT * FROM V$TRANSPORTABLE_PLATFORM;

PLATFORM_ID PLATFORM_NAME                  ENDIAN_FORMAT
----------- ------------------------------ --------------
          1 Solaris[tm] OE (32-bit)        Big
          2 Solaris[tm] OE (64-bit)        Big
          7 Microsoft Windows NT           Little
```

```
         10 Linux IA (32-bit)              Little
          6 AIX-Based Systems (64-bit)     Big
          3 HP-UX (64-bit)                 Big
          5 HP Tru64 UNIX                  Little
          4 HP-UX IA (64-bit)              Big
         11 Linux IA (64-bit)              Little
         15 HP Open VMS                    Little

10 rows selected.
```

If the source platform and the target platform are of different endianness, then an additional step must be done on either the source or target platform to convert the tablespace being transported to the target format. If they are of the same endianness, then no conversion is necessary and tablespaces can be transported as if they were on the same platform.

Before a tablespace can be transported to a different platform, the datafile header must identify the platform to which it belongs. In an Oracle Database with compatibility set to 10.0.0 or higher, you can accomplish this by making the datafile read / write at least once.

Limitations on Transportable Tablespace Use

Be aware of the following limitations as you plan to transport tablespaces:

- The source and target database must use the same character set and national character set.
- You cannot transport a tablespace to a target database in which a tablespace with the same name already exists. However, you can rename either the tablespace to be transported or the destination tablespace before the transport operation.
- Objects with underlying objects (such as materialized views) or contained objects (such as partitioned tables) are not transportable unless all of the underlying or contained objects are in the tablespace set.

Most database entities, such as data in a tablespace or structural information associated with the tablespace, behave normally after being transported to a different database. Some exceptions arise with the following entities:

Advanced Queues Transportable tablespaces do not support 8.0-compatible advanced queues with multiple recipients.

SYSTEM Tablespace Objects You cannot transport the SYSTEM tablespace or objects owned by the user SYS. Some examples of such objects are PL/SQL, Java classes, callouts, views, synonyms, users, privileges, dimensions, directories, and sequences.

Opaque Types Types whose interpretation is application-specific and opaque to the database (such as `RAW`, `BFILE`, and the AnyTypes) can be transported, but they are not converted as part of the cross-platform transport operation. Their actual structure is known only to the application, so the application must address any endianness issues after these types are moved to the new platform. Types and objects that use these opaque types, either directly or indirectly, are also subject to this limitation.

Floating-Point Numbers `BINARY_FLOAT` and `BINARY_DOUBLE` types are transportable using Data Pump but not using the original export utility.

Compatibility Considerations for Transportable Tablespaces

When you create a transportable tablespace set, Oracle Database computes the lowest compatibility level at which the target database must run. This is referred to as the compatibility level of the transportable set. Beginning with Oracle Database 10*g*, a tablespace can always be transported to a database with the same or higher compatibility setting, whether the target database is on the same or a different platform. The database signals an error if the compatibility level of the transportable set is higher than the compatibility level of the target database.

The following table shows the minimum compatibility requirements of the source and target tablespace in various scenarios. The source and target database need not have the same compatibility setting.

Table 8–1 Minimum Compatibility Requirements

	Minimum Compatibility Setting	
Transport Scenario	**Source Database**	**Target Database**
Databases on the same platform	8.0	8.0
Tablespace with different database block size than the target database	9.0	9.0
Databases on different platforms	10.0	10.0

Transporting Tablespaces Between Databases: A Procedure and Example

To move or copy a set of tablespaces, perform the following steps.

1. For cross-platform transport, check the endian format of both platforms by querying the V$TRANSPORTABLE_PLATFORM view.

   If you are transporting the tablespace set to a platform different from the source platform, then determine if the source and target platforms are supported and their endianness. If both platforms have the same endianness, no conversion is necessary. Otherwise you must do a conversion of the tablespace set either at the source or target database.

   Ignore this step if you are transporting your tablespace set to the same platform.

2. Pick a self-contained set of tablespaces.

3. Generate a transportable tablespace set.

   A transportable tablespace set consists of datafiles for the set of tablespaces being transported and an export file containing structural information for the set of tablespaces.

   If you are transporting the tablespace set to a platform with different endianness from the source platform, you must convert the tablespace set to the endianness of the target platform. You can perform a source-side conversion at this step in the procedure, or you can perform a target-side conversion as part of step 4.

4. Transport the tablespace set.

   Copy the datafiles and the export file to the target database. You can do this using any facility for copying flat files (for example, an operating system copy utility, ftp, the DBMS_FILE_COPY package, or publishing on CDs).

   If you have transported the tablespace set to a platform with different endianness from the source platform, and you have not performed a source-side conversion to the endianness of the target platform, you should perform a target-side conversion now.

5. Plug in the tablespace.

   Invoke the Data Pump utility to plug the set of tablespaces into the target database.

These steps are illustrated more fully in the example that follows, where it is assumed the following datafiles and tablespaces exist:

Tablespace	Datafile:
sales_1	/u01/oracle/oradata/salesdb/sales_101.dbf
sales_2	/u01/oracle/oradata/salesdb/sales_201.dbf

Step 1: Determine if Platforms are Supported and Endianness

This step is only necessary if you are transporting the tablespace set to a platform different from the source platform. If sales_1 and sales_2 were being transported to a different platform, you can execute the following query on both platforms to determine if the platforms are supported and their endian formats:

```
SELECT d.PLATFORM_NAME, ENDIAN_FORMAT
     FROM V$TRANSPORTABLE_PLATFORM tp, V$DATABASE d
     WHERE tp.PLATFORM_NAME = d.PLATFORM_NAME;
```

The following is the query result from the source platform:

```
PLATFORM_NAME             ENDIAN_FORMAT
------------------------- --------------
Solaris[tm] OE (32-bit)   Big
```

The following is the result from the target platform:

```
PLATFORM_NAME             ENDIAN_FORMAT
------------------------- --------------
Microsoft Windows NT      Little
```

You can see that the endian formats are different and thus a conversion is necessary for transporting the tablespace set.

Step 2: Pick a Self-Contained Set of Tablespaces

There may be logical or physical dependencies between objects in the transportable set and those outside of the set. You can only transport a set of tablespaces that is self-contained. In this context "self-contained" means that there are no references from inside the set of tablespaces pointing outside of the tablespaces. Some examples of self contained tablespace violations are:

- An index inside the set of tablespaces is for a table outside of the set of tablespaces.

> **Note:** It is not a violation if a corresponding index for a table is outside of the set of tablespaces.

- A partitioned table is partially contained in the set of tablespaces.

  The tablespace set you want to copy must contain either all partitions of a partitioned table, or none of the partitions of a partitioned table. If you want to transport a subset of a partition table, you must exchange the partitions into tables.

- A referential integrity constraint points to a table across a set boundary.

  When transporting a set of tablespaces, you can choose to include referential integrity constraints. However, doing so can affect whether or not a set of tablespaces is self-contained. If you decide not to transport constraints, then the constraints are not considered as pointers.

- A table inside the set of tablespaces contains a `LOB` column that points to `LOB`s outside the set of tablespaces.

To determine whether a set of tablespaces is self-contained, you can invoke the `TRANSPORT_SET_CHECK` procedure in the Oracle supplied package `DBMS_TTS`. You must have been granted the `EXECUTE_CATALOG_ROLE` role (initially signed to `SYS`) to execute this procedure.

When you invoke the `DBMS_TTS` package, you specify the list of tablespaces in the transportable set to be checked for self containment. You can optionally specify if constraints must be included. For strict or full containment, you must additionally set the `TTS_FULL_CHECK` parameter to `TRUE`.

The strict or full containment check is for cases that require capturing not only references going outside the transportable set, but also those coming into the set. Tablespace Point-in-Time Recovery (TSPITR) is one such case where dependent objects must be fully contained or fully outside the transportable set.

For example, it is a violation to perform TSPITR on a tablespace containing a table `t` but not its index `i` because the index and data will be inconsistent after the transport. A full containment check ensures that there are no dependencies going outside or coming into the transportable set. See the example for TSPITR in the *Oracle Database Backup and Recovery Advanced User's Guide*.

Note: The default for transportable tablespaces is to check for self containment rather than full containment.

The following statement can be used to determine whether tablespaces sales_1 and sales_2 are self-contained, with referential integrity constraints taken into consideration (indicated by TRUE).

```
EXECUTE DBMS_TTS.TRANSPORT_SET_CHECK('sales_1,sales_2', TRUE);
```

After invoking this PL/SQL package, you can see all violations by selecting from the TRANSPORT_SET_VIOLATIONS view. If the set of tablespaces is self-contained, this view is empty. The following example illustrates a case where there are two violations: a foreign key constraint, dept_fk, across the tablespace set boundary, and a partitioned table, jim.sales, that is partially contained in the tablespace set.

```
SQL> SELECT * FROM TRANSPORT_SET_VIOLATIONS;

VIOLATIONS
--------------------------------------------------------------------------------
Constraint DEPT_FK between table JIM.EMP in tablespace SALES_1 and table
JIM.DEPT in tablespace OTHER
Partitioned table JIM.SALES is partially contained in the transportable set
```

These violations must be resolved before sales_1 and sales_2 are transportable. As noted in the next step, one choice for bypassing the integrity constraint violation is to not export the integrity constraints.

See Also:

- *PL/SQL Packages and Types Reference* for more information about the DBMS_TTS package
- *Oracle Database Backup and Recovery Advanced User's Guide* for information specific to using the DBMS_TTS package for TSPITR

Step 3: Generate a Transportable Tablespace Set

Any privileged user can perform this step. However, you must have been assigned the EXP_FULL_DATABASE role to perform a transportable tablespace export operation.

After ensuring you have a self-contained set of tablespaces that you want to transport, generate a transportable tablespace set by performing the following actions:

1. Make all tablespaces in the set you are copying read-only.

```
SQL> ALTER TABLESPACE sales_1 READ ONLY;

Tablespace altered.

SQL> ALTER TABLESPACE sales_2 READ ONLY;

Tablespace altered.
```

2. Invoke the Data Pump export utility on the host system and specify which tablespaces are in the transportable set.

```
SQL> HOST

$ EXPDP system/password DUMPFILE=expdat.dmp DIRECTORY=dpump_dir
        TRANSPORT_TABLESPACES = sales_1,sales_2
```

   You must always specify TRANSPORT_TABLESPACES, which determines the mode of the export operation. In this example:

   - The DUMPFILE parameter specifies the name of the structural information export file to be created, expdat.dmp.
   - The DIRECTORY parameter specifies the default directory object that points to the operating system location of the dump file. You must create the DIRECTORY object before invoking Data Pump, and you must grant REWRITE object privilege on the directory to PUBLIC.
   - Triggers and indexes are included in the export operation by default.

   If you want to perform a transport tablespace operation with a strict containment check, use the TRANSPORT_FULL_CHECK parameter, as shown in the following example:

```
EXPDP system/password DUMPFILE=expdat.dmp DIRECTORY = dpump_dir
      TRANSPORT_TABLESPACES=sales_1,sales_2 TRANSPORT_FULL_CHECK=Y
```

   In this example, the Data Pump export utility verifies that there are no dependencies between the objects inside the transportable set and objects outside the transportable set. If the tablespace set being transported is not

self-contained, then the export fails and indicates that the transportable set is not self-contained. You must then return to Step 1 to resolve all violations.

Notes: The Data Pump utility is used to export only data dictionary structural information (metadata) for the tablespaces. No actual data is unloaded, so this operation goes relatively quickly even for large tablespace sets.

---

3. When finished, exit back to SQL*Plus:

```
$ EXIT
```

See Also: *Oracle Database Utilities* for information about using the Data Pump utility

If `sales_1` and `sales_2` are being transported to a different platform, and the endianness of the platforms is different, and if you want to convert before transporting the tablespace set, then convert the datafiles composing the `sales_1` and `sales_2` tablespaces:

4. From SQL*Plus, return to the host system:

```
SQL> HOST
```

5. The RMAN `CONVERT` command is used to do the conversion. Connect to RMAN:

```
$ RMAN TARGET /

Recovery Manager: Release 10.1.0.0.0

Copyright (c) 1995, 2003, Oracle Corporation.  All rights reserved.

connected to target database: salesdb (DBID=3295731590)
```

6. Convert the datafiles into a temporary location on the source platform. In this example, assume that the temporary location, directory `/temp`, has already been created. The converted datafiles are assigned names by the system.

```
RMAN> CONVERT TABLESPACE sales_1,sales_2
2> TO PLATFORM 'Microsoft Windows NT'
3> FORMAT '/temp/%U';

Starting backup at 08-APR-03
```

```
using target database control file instead of recovery catalog
allocated channel: ORA_DISK_1
channel ORA_DISK_1: sid=11 devtype=DISK
channel ORA_DISK_1: starting datafile conversion
input datafile fno=00005 name=/u01/oracle/oradata/salesdb/sales_101.dbf
converted datafile=/temp/data_D-10_I-3295731590_TS-ADMIN_TBS_FNO-5_05ek24v5
channel ORA_DISK_1: datafile conversion complete, elapsed time: 00:00:15
channel ORA_DISK_1: starting datafile conversion
input datafile fno=00004 name=/u01/oracle/oradata/salesdb/sales_101.dbf
converted datafile=/temp/data_D-10_I-3295731590_TS-EXAMPLE_FNO-4_06ek24vl
channel ORA_DISK_1: datafile conversion complete, elapsed time: 00:00:45
Finished backup at 08-APR-03
```

> **See Also:** *Oracle Database Recovery Manager Reference* for a description of the RMAN CONVERT command

7. Exit recovery manager:

```
RMAN> exit
Recovery Manager complete.
```

Step 4: Transport the Tablespace Set

Transport *both the datafiles and the export file* of the tablespaces to a place accessible to the target database. You can use any facility for copying flat files (for example, an operating system copy utility, ftp, the DBMS_FILE_TRANSFER package, or publishing on CDs).

> **Caution:** Exercise caution when using the UNIX dd utility to copy raw-device files between databases. The dd utility can be used to copy an entire source raw-device file, or it can be invoked with options that instruct it to copy only a specific range of blocks from the source raw-device file.
>
> It is difficult to ascertain actual datafile size for a raw-device file because of hidden control information that is stored as part of the datafile. Thus, it is advisable when using the dd utility to specify copying the entire source raw-device file contents.

If you are transporting the tablespace set to a platform with endianness that is different from the source platform, and you have not yet converted the tablespace

set, you must do so now. This example assumes that you have completed the following steps before the transport:

1. Set the source tablespaces to be transported to be read-only.
2. Use the export utility to create an export file (in our example, expdat.dmp).

Datafiles that are to be converted on the target platform can be moved to a temporary location on the target platform. However, all datafiles, whether already converted or not, must be moved to a designated location on the target database.

Now use RMAN to convert the necessary transported datafiles to the endian format of the destination host format and deposit the results in /orahome/dbs, as shown in this hypothetical example:

```
RMAN> CONVERT DATAFILE
2> '/hq/finance/work/tru/tbs_31.f',
3> '/hq/finance/work/tru/tbs_32.f',
4> '/hq/finance/work/tru/tbs_41.f'
5> TO PLATFORM="Solaris[tm] OE (32-bit)"
6> FROM PLATFORM="HP TRu64 UNIX"
7> DBFILE_NAME_CONVERT=
8> "/hq/finance/work/tru/", "/hq/finance/dbs/tru"
9> PARALLELISM=5;
```

You identify the datafiles by filename, not by tablespace name. Until the datafiles are plugged in, the local instance has no way of knowing the desired tablespace names. The source and destination platforms are optional. RMAN determines the source platform by examining the datafile, and the target platform defaults to the platform of the host running the conversion.

> **See Also:** "Copying Files Using the Database Server" on page 9-15 for information about using the DBMS_FILE_TRANSFER package to copy the files that are being transported and their metadata

Step 5: Plug In the Tablespace Set

> **Note:** If you are transporting a tablespace of a different block size than the standard block size of the database receiving the tablespace set, then you must first have a DB_*n*K_CACHE_SIZE initialization parameter entry in the receiving database parameter file.
>
> For example, if you are transporting a tablespace with an 8K block size into a database with a 4K standard block size, then you must include a DB_8K_CACHE_SIZE initialization parameter entry in the parameter file. If it is not already included in the parameter file, this parameter can be set using the ALTER SYSTEM SET statement.
>
> See *Oracle Database Reference* for information about specifying values for the DB_*n*K_CACHE_SIZE initialization parameter.

Any privileged user can perform this step. To plug in a tablespace set, perform the following tasks:

1. Plug in the tablespaces and integrate the structural information using the Data Pump Import utility, impdp:

```
IMPDP system/password DUMPFILE=expdat.dmp DIRECTORY=dpump_dir
   TRANSPORT_DATAFILES=
   /salesdb/sales_101.dbf,
   /salesdb/sales_201.dbf
   REMAP_SCHEMA=(dcranney:smith) REMAP_SCHEMA=(jfee:williams)
```

   In this example we specify the following:

   - The DUMPFILE parameter specifies the exported file containing the metadata for the tablespaces to be imported.
   - The DIRECTORY parameter specifies the directory object that identifies the location of the dump file.
   - The TRANSPORT_DATAFILES parameter identifies all of the datafiles containing the tablespaces to be imported.
   - The REMAP_SCHEMA parameter changes the ownership of database objects. If you do not specify REMAP_SCHEMA, all database objects (such as tables and indexes) are created in the same user schema as in the source database, and those users must already exist in the target database. If they do not

exist, then the import utility returns an error. In this example, objects in the tablespace set owned by `dcranney` in the source database will be owned by `smith` in the target database after the tablespace set is plugged in. Similarly, objects owned by `jfee` in the source database will be owned by `williams` in the target database. In this case, the target database is not required to have users `dcranney` and `jfee`, but must have users `smith` and `williams`.

After this statement executes successfully, all tablespaces in the set being copied remain in read-only mode. Check the import logs to ensure that no error has occurred.

When dealing with a large number of datafiles, specifying the list of datafile names in the statement line can be a laborious process. It can even exceed the statement line limit. In this situation, you can use an import parameter file. For example, you can invoke the Data Pump import utility as follows:

```
IMPDP system/password PARFILE='par.f'
```

where the parameter file, `par.f` contains the following:

```
DIRECTORY=dpump_dir
DUMPFILE=expdat.dmp
TRANSPORT_DATAFILES="'/db/sales_jan','/db/sales_feb'"
REMAP_SCHEMA=dcranney:smith
REMAP_SCHEMA=jfee:williams
```

See Also: *Oracle Database Utilities* for information about using the import utility

2. If required, put the tablespaces into read / write mode as follows:

```
ALTER TABLESPACE sales_1 READ WRITE;
ALTER TABLESPACE sales_2 READ WRITE;
```

Using Transportable Tablespaces: Scenarios

The following sections describe some uses for transportable tablespaces:

- Transporting and Attaching Partitions for Data Warehousing
- Publishing Structured Data on CDs
- Mounting the Same Tablespace Read-Only on Multiple Databases
- Archiving Historical Data Using Transportable Tablespaces

- Using Transportable Tablespaces to Perform TSPITR

Transporting and Attaching Partitions for Data Warehousing

Typical enterprise data warehouses contain one or more large fact tables. These fact tables can be partitioned by date, making the enterprise data warehouse a historical database. You can build indexes to speed up star queries. Oracle recommends that you build local indexes for such historically partitioned tables to avoid rebuilding global indexes every time you drop the oldest partition from the historical database.

Suppose every month you would like to load one month of data into the data warehouse. There is a large fact table in the data warehouse called `sales`, which has the following columns:

```
CREATE TABLE sales (invoice_no NUMBER,
   sale_year  INT NOT NULL,
   sale_month INT NOT NULL,
   sale_day   INT NOT NULL)
   PARTITION BY RANGE (sale_year, sale_month, sale_day)
     (partition jan98 VALUES LESS THAN (1998, 2, 1),
      partition feb98 VALUES LESS THAN (1998, 3, 1),
      partition mar98 VALUES LESS THAN (1998, 4, 1),
      partition apr98 VALUES LESS THAN (1998, 5, 1),
      partition may98 VALUES LESS THAN (1998, 6, 1),
      partition jun98 VALUES LESS THAN (1998, 7, 1));
```

You create a local nonprefixed index:

```
CREATE INDEX sales_index ON sales(invoice_no) LOCAL;
```

Initially, all partitions are empty, and are in the same default tablespace. Each month, you want to create one partition and attach it to the partitioned `sales` table.

Suppose it is July 1998, and you would like to load the July sales data into the partitioned table. In a staging database, you create a new tablespace, `ts_jul`. You also create a table, `jul_sales`, in that tablespace with exactly the same column types as the `sales` table. You can create the table `jul_sales` using the CREATE TABLE ... AS SELECT statement. After creating and populating `jul_sales`, you can also create an index, `jul_sale_index`, for the table, indexing the same column as the local index in the `sales` table. After building the index, transport the tablespace `ts_jul` to the data warehouse.

In the data warehouse, add a partition to the `sales` table for the July sales data. This also creates another partition for the local nonprefixed index:

```
ALTER TABLE sales ADD PARTITION jul98 VALUES LESS THAN (1998, 8, 1);
```

Attach the transported table jul_sales to the table sales by exchanging it with the new partition:

```
ALTER TABLE sales EXCHANGE PARTITION jul98 WITH TABLE jul_sales
   INCLUDING INDEXES
   WITHOUT VALIDATION;
```

This statement places the July sales data into the new partition jul98, attaching the new data to the partitioned table. This statement also converts the index jul_sale_index into a partition of the local index for the sales table. This statement should return immediately, because it only operates on the structural information and it simply switches database pointers. If you know that the data in the new partition does not overlap with data in previous partitions, you are advised to specify the WITHOUT VALIDATION clause. Otherwise, the statement goes through all the new data in the new partition in an attempt to validate the range of that partition.

If all partitions of the sales table came from the same staging database (the staging database is never destroyed), the exchange statement always succeeds. In general, however, if data in a partitioned table comes from different databases, it is possible that the exchange operation may fail. For example, if the jan98 partition of sales did not come from the same staging database, the preceding exchange operation can fail, returning the following error:

```
ORA-19728: data object number conflict between table JUL_SALES and partition
JAN98 in table SALES
```

To resolve this conflict, move the offending partition by issuing the following statement:

```
ALTER TABLE sales MOVE PARTITION jan98;
```

Then retry the exchange operation.

After the exchange succeeds, you can safely drop jul_sales and jul_sale_index (both are now empty). Thus you have successfully loaded the July sales data into your data warehouse.

Publishing Structured Data on CDs

Transportable tablespaces provide a way to publish structured data on CDs. A data provider can load a tablespace with data to be published, generate the transportable set, and copy the transportable set to a CD. This CD can then be distributed.

When customers receive this CD, they can plug it into an existing database without having to copy the datafiles from the CD to disk storage. For example, suppose on a Windows NT machine D: drive is the CD drive. You can plug in a transportable set with datafile `catalog.f` and export file `expdat.dmp` as follows:

```
IMPDP system/password DUMPFILE=expdat.dmp DIRECTORY=dpump_dir
   TRANSPORT_DATAFILES='D:\catalog.f'
```

You can remove the CD while the database is still up. Subsequent queries to the tablespace return an error indicating that the database cannot open the datafiles on the CD. However, operations to other parts of the database are not affected. Placing the CD back into the drive makes the tablespace readable again.

Removing the CD is the same as removing the datafiles of a read-only tablespace. If you shut down and restart the database, the database indicates that it cannot find the removed datafile and does not open the database (unless you set the initialization parameter READ_ONLY_OPEN_DELAYED to TRUE). When READ_ONLY_OPEN_DELAYED is set to TRUE, the database reads the file only when someone queries the plugged-in tablespace. Thus, when plugging in a tablespace on a CD, you should always set the READ_ONLY_OPEN_DELAYED initialization parameter to TRUE, unless the CD is permanently attached to the database.

Mounting the Same Tablespace Read-Only on Multiple Databases

You can use transportable tablespaces to mount a tablespace read-only on multiple databases. In this way, separate databases can share the same data on disk instead of duplicating data on separate disks. The tablespace datafiles must be accessible by all databases. To avoid database corruption, the tablespace must remain read-only in all the databases mounting the tablespace.

You can mount the same tablespace read-only on multiple databases in either of the following ways:

- Plug the tablespace into each of the databases on which you want to mount the tablespace. Generate a transportable set in a single database. Put the datafiles in the transportable set on a disk accessible to all databases. Import the structural information into each database.
- Generate the transportable set in one of the databases and plug it into other databases. If you use this approach, it is assumed that the datafiles are already on the shared disk, and they belong to an existing tablespace in one of the databases. You can make the tablespace read-only, generate the transportable set, and then plug the tablespace in to other databases while the datafiles remain in the same location on the shared disk.

You can make the disk accessible by multiple computers in several ways. You can use either a cluster file system or raw disk. You can also use network file system (NFS), but be aware that if a user queries the shared tablespace while NFS is down, the database will hang until the NFS operation times out.

Later, you can drop the read-only tablespace in some of the databases. Doing so does not modify the datafiles for the tablespace. Thus, the drop operation does not corrupt the tablespace. Do not make the tablespace read/write unless only one database is mounting the tablespace.

Archiving Historical Data Using Transportable Tablespaces

Since a transportable tablespace set is a self-contained set of files that can be plugged into any Oracle Database, you can archive old/historical data in an enterprise data warehouse using the transportable tablespace procedures described in this chapter.

> **See Also:** *Oracle Data Warehousing Guide* for more details

Using Transportable Tablespaces to Perform TSPITR

You can use transportable tablespaces to perform tablespace point-in-time recovery (TSPITR).

> **See Also:** *Oracle Database Backup and Recovery Advanced User's Guide* for information about how to perform TSPITR using transportable tablespaces

Moving Databases Across Platforms Using Transportable Tablespaces

You can use the transportable tablespace feature to migrate a database to a different platform by creating a new database on the destination platform and performing a transport of all the user tablespaces.

You cannot transport the SYSTEM tablespace. Therefore, objects such as sequences, PL/SQL packages, and other objects that depend on the SYSTEM tablespace are not transported. You must either create these objects manually on the destination database, or use Data Pump to transport the objects that are not moved by transportable tablespace.

Viewing Tablespace Information

The following data dictionary and dynamic performance views provide useful information about the tablespaces of a database.

View	Description
V$TABLESPACE	Name and number of all tablespaces from the control file.
DBA_TABLESPACES, USER_TABLESPACES	Descriptions of all (or user accessible) tablespaces.
DBA_TABLESPACE_GROUPS	Displays the tablespace groups and the tablespaces that belong to them.
DBA_SEGMENTS, USER_SEGMENTS	Information about segments within all (or user accessible) tablespaces.
DBA_EXTENTS, USER_EXTENTS	Information about data extents within all (or user accessible) tablespaces.
DBA_FREE_SPACE, USER_FREE_SPACE	Information about free extents within all (or user accessible) tablespaces.
V$DATAFILE	Information about all datafiles, including tablespace number of owning tablespace.
V$TEMPFILE	Information about all tempfiles, including tablespace number of owning tablespace.
DBA_DATA_FILES	Shows files (datafiles) belonging to tablespaces.
DBA_TEMP_FILES	Shows files (tempfiles) belonging to temporary tablespaces.
V$TEMP_EXTENT_MAP	Information for all extents in all locally managed temporary tablespaces.
V$TEMP_EXTENT_POOL	For locally managed temporary tablespaces: the state of temporary space cached and used for by each instance.
V$TEMP_SPACE_HEADER	Shows space used / free for each tempfile.
DBA_USERS	Default and temporary tablespaces for all users.
DBA_TS_QUOTAS	Lists tablespace quotas for all users.
V$SORT_SEGMENT	Information about every sort segment in a given instance. The view is only updated when the tablespace is of the TEMPORARY type.
V$TEMPSEG_USAGE	Describes temporary (sort) segment usage by user for temporary or permanent tablespaces.

The following are just a few examples of using some of these views.

> **See Also:** *Oracle Database Reference* for complete description of these views

Example 1: Listing Tablespaces and Default Storage Parameters

To list the names and default storage parameters of all tablespaces in a database, use the following query on the DBA_TABLESPACES view:

```
SELECT TABLESPACE_NAME "TABLESPACE",
   INITIAL_EXTENT "INITIAL_EXT",
   NEXT_EXTENT "NEXT_EXT",
   MIN_EXTENTS "MIN_EXT",
   MAX_EXTENTS "MAX_EXT",
   PCT_INCREASE
   FROM DBA_TABLESPACES;

TABLESPACE  INITIAL_EXT  NEXT_EXT  MIN_EXT   MAX_EXT    PCT_INCREASE
----------  -----------  --------  -------   -------    ------------
RBS             1048576   1048576        2        40               0
SYSTEM           106496    106496        1        99               1
TEMP             106496    106496        1        99               0
TESTTBS           57344     16384        2        10               1
USERS             57344     57344        1        99               1
```

Example 2: Listing the Datafiles and Associated Tablespaces of a Database

To list the names, sizes, and associated tablespaces of a database, enter the following query on the DBA_DATA_FILES view:

```
SELECT  FILE_NAME, BLOCKS, TABLESPACE_NAME
   FROM DBA_DATA_FILES;

FILE_NAME                                    BLOCKS  TABLESPACE_NAME
-----------                                ----------  ------------------
/U02/ORACLE/IDDB3/DBF/RBS01.DBF                1536  RBS
/U02/ORACLE/IDDB3/DBF/SYSTEM01.DBF             6586  SYSTEM
/U02/ORACLE/IDDB3/DBF/TEMP01.DBF               6400  TEMP
/U02/ORACLE/IDDB3/DBF/TESTTBS01.DBF            6400  TESTTBS
/U02/ORACLE/IDDB3/DBF/USERS01.DBF               384  USERS
```

Example 3: Displaying Statistics for Free Space (Extents) of Each Tablespace

To produce statistics about free extents and coalescing activity for each tablespace in the database, enter the following query:

```
SELECT TABLESPACE_NAME "TABLESPACE", FILE_ID,
   COUNT(*)    "PIECES",
   MAX(blocks) "MAXIMUM",
   MIN(blocks) "MINIMUM",
   AVG(blocks) "AVERAGE",
   SUM(blocks) "TOTAL"
   FROM DBA_FREE_SPACE
GROUP BY TABLESPACE_NAME, FILE_ID;

TABLESPACE    FILE_ID  PIECES   MAXIMUM    MINIMUM  AVERAGE    TOTAL
----------    -------  ------   -------    -------  -------   ------
RBS                 2       1       955        955      955       955
SYSTEM              1       1       119        119      119       119
TEMP                4       1      6399       6399     6399      6399
TESTTBS             5       5      6364          3     1278      6390
USERS               3       1       363        363      363       363
```

PIECES shows the number of free space extents in the tablespace file, MAXIMUM and MINIMUM show the largest and smallest contiguous area of space in database blocks, AVERAGE shows the average size in blocks of a free space extent, and TOTAL shows the amount of free space in each tablespace file in blocks. This query is useful when you are going to create a new object or you know that a segment is about to extend, and you want to make sure that there is enough space in the containing tablespace.

9

Managing Datafiles and Tempfiles

This chapter describes the various aspects of datafile and tempfile management, and contains the following topics:

- Guidelines for Managing Datafiles
- Creating Datafiles and Adding Datafiles to a Tablespace
- Changing Datafile Size
- Altering Datafile Availability
- Renaming and Relocating Datafiles
- Dropping Datafiles
- Verifying Data Blocks in Datafiles
- Copying Files Using the Database Server
- Mapping Files to Physical Devices
- Viewing Datafile Information

> **See Also:** Part III, "Automated File and Storage Management" for information about creating datafiles and tempfiles that are both created and managed by the Oracle Database server

Guidelines for Managing Datafiles

Datafiles are physical files of the operating system that store the data of all logical structures in the database. They must be explicitly created for each tablespace.

> **Note:** Tempfiles are a special class of datafiles that are associated only with temporary tablespaces. Information is this chapter applies to both datafiles and tempfiles except where differences are noted. Tempfiles are further described in "Creating a Locally Managed Temporary Tablespace" on page 8-18

Oracle Database assigns each datafile two associated file numbers, an absolute file number and a relative file number, that are used to uniquely identify it. These numbers are described in the following table:

Type of File Number	Description
Absolute	Uniquely identifies a datafile *in the database*. This file number can be used in many SQL statements that reference datafiles in place of using the file name. The absolute file number can be found in the `FILE#` column of the `V$DATAFILE` or `V$TEMPFILE` view, or in the `FILE_ID` column of the `DBA_DATA_FILES` or `DBA_TEMP_FILES` view.
Relative	Uniquely identifies a datafile *within a tablespace*. For small and medium size databases, relative file numbers usually have the same value as the absolute file number. However, when the number of datafiles in a database exceeds a threshold (typically 1023), the relative file number differs from the absolute file number. In a bigfile tablespace, the relative file number is always 1024 (4096 on OS/390 platform).

This section describes aspects of managing datafiles, and contains the following topics:

- Determine the Number of Datafiles
- Determine the Size of Datafiles
- Place Datafiles Appropriately
- Store Datafiles Separate from Redo Log Files

Determine the Number of Datafiles

At least one datafile is required for the `SYSTEM` and `SYSAUX` tablespaces of a database. Your database should contain several other tablespaces with their associated datafiles or tempfiles. The number of datafiles that you anticipate

creating for your database can affect the settings of initialization parameters and the specification of CREATE DATABASE statement clauses.

Be aware that your operating system might impose limits on the number of datafiles contained in your Oracle Database. Also consider that the number of datafiles, and how and where they are allocated can affect the performance of your database.

> **Note:** One means of controlling the number of datafiles in your database and simplifying their management is to use bigfile tablespaces. Bigfile tablespaces comprise a single, very large datafile and are especially useful in ultra large databases and where a logical volume manager is used for managing operating system files. Bigfile tablespaces are discussed in "Bigfile Tablespaces" on page 8-9.

Consider the following guidelines when determining the number of datafiles for your database.

Determine a Value for the DB_FILES Initialization Parameter

When starting an Oracle Database instance, the DB_FILES initialization parameter indicates the amount of SGA space to reserve for datafile information and thus, the maximum number of datafiles that can be created for the instance. This limit applies for the life of the instance. You can change the value of DB_FILES (by changing the initialization parameter setting), but the new value does not take effect until you shut down and restart the instance.

When determining a value for DB_FILES, take the following into consideration:

- If the value of DB_FILES is too low, you cannot add datafiles beyond the DB_FILES limit without first shutting down the database.
- If the value of DB_FILES is too high, memory is unnecessarily consumed.

Consider Possible Limitations When Adding Datafiles to a Tablespace

You can add datafiles to traditional smallfile tablespaces, subject to the following limitations:

- Operating systems often impose a limit on the number of files a process can open simultaneously. More datafiles cannot be created when the operating system limit of open files is reached.

- Operating systems impose limits on the number and size of datafiles.
- The database imposes a maximum limit on the number of datafiles for any Oracle Database opened by any instance. This limit is operating system specific.
- You cannot exceed the number of datafiles specified by the `DB_FILES` initialization parameter.
- When you issue `CREATE DATABASE` or `CREATE CONTROLFILE` statements, the `MAXDATAFILES` parameter specifies an initial size of the datafile portion of the control file. However, if you attempt to add a new file whose number is greater than `MAXDATAFILES`, but less than or equal to `DB_FILES`, the control file will expand automatically so that the datafiles section can accommodate more files.

Consider the Performance Impact

The number of datafiles contained in a tablespace, and ultimately the database, can have an impact upon performance.

Oracle Database allows more datafiles in the database than the operating system defined limit. The database DBW*n* processes can open all online datafiles. Oracle Database is capable of treating open file descriptors as a cache, automatically closing files when the number of open file descriptors reaches the operating system-defined limit. This can have a negative performance impact. When possible, adjust the operating system limit on open file descriptors so that it is larger than the number of online datafiles in the database.

See Also:

- Your operating system specific Oracle documentation for more information on operating system limits
- *Oracle Database SQL Reference* for more information about the `MAXDATAFILES` parameter of the `CREATE DATABASE` or `CREATE CONTROLFILE` statement

Determine the Size of Datafiles

When creating a tablespace, you should estimate the potential size of database objects and create sufficient datafiles. Later, if needed, you can create additional datafiles and add them to a tablespace to increase the total amount of disk space allocated to it, and consequently the database. Preferably, place datafiles on multiple devices to ensure that data is spread evenly across all devices.

Place Datafiles Appropriately

Tablespace location is determined by the physical location of the datafiles that constitute that tablespace. Use the hardware resources of your computer appropriately.

For example, if several disk drives are available to store the database, consider placing potentially contending datafiles on separate disks.This way, when users query information, both disk drives can work simultaneously, retrieving data at the same time.

> **See Also:** *Oracle Database Performance Tuning Guide* for information about I/O and the placement of datafiles

Store Datafiles Separate from Redo Log Files

Datafiles should not be stored on the same disk drive that stores the database redo log files. If the datafiles and redo log files are stored on the same disk drive and that disk drive fails, the files cannot be used in your database recovery procedures.

If you multiplex your redo log files, then the likelihood of losing all of your redo log files is low, so you can store datafiles on the same drive as some redo log files.

Creating Datafiles and Adding Datafiles to a Tablespace

You can create datafiles and associate them with a tablespace using any of the statements listed in the following table. In all cases, you can either specify the file specifications for the datafiles being created, or you can use the Oracle-managed files feature to create files that are created and managed by the database server. The table includes a brief description of the statement, as used to create datafiles, and references the section of this book where use of the statement is specifically described:

SQL Statement	Description	Additional Information
`CREATE TABLESPACE`	Creates a tablespace and the datafiles that comprise it	"Creating Tablespaces" on page 8-3
`CREATE TEMPORARY TABLESPACE`	Creates a locally-managed temporary tablespace and the *tempfiles* (tempfiles are a special kind of datafile) that comprise it	"Creating a Locally Managed Temporary Tablespace" on page 8-18

SQL Statement	Description	Additional Information
`ALTER TABLESPACE ... ADD DATAFILE`	Creates and adds a datafile to a tablespace	"Altering a Dictionary-Managed Tablespace" on page 8-13 and "Altering a Locally Managed Temporary Tablespace" on page 8-19
`ALTER TABLESPACE ... ADD TEMPFILE`	Creates and adds a tempfile to a temporary tablespace	"Creating a Locally Managed Temporary Tablespace" on page 8-18
`CREATE DATABASE`	Creates a database and associated datafiles	"Manually Creating an Oracle Database" on page 2-2
`ALTER DATABASE ... CREATE DATAFILE`	Creates a new empty datafile in place of an old one--useful to re-create a datafile that was lost with no backup.	See *Oracle Database Backup and Recovery Advanced User's Guide*.

If you add new datafiles to a tablespace and do not fully specify the filenames, the database creates the datafiles in the default database directory or the current directory, depending upon your operating system. Oracle recommends you always specify a fully qualified name for a datafile. Unless you want to reuse existing files, make sure the new filenames do not conflict with other files. Old files that have been previously dropped will be overwritten.

If a statement that creates a datafile fails, the database removes any created operating system files. However, because of the large number of potential errors that can occur with file systems and storage subsystems, there can be situations where you must manually remove the files using operating system commands.

Changing Datafile Size

This section describes the various ways to alter the size of a datafile, and contains the following topics:

- Enabling and Disabling Automatic Extension for a Datafile
- Manually Resizing a Datafile

Enabling and Disabling Automatic Extension for a Datafile

You can create datafiles or alter existing datafiles so that they automatically increase in size when more space is needed in the database. The file size increases in specified increments up to a specified maximum.

Setting your datafiles to extend automatically provides these advantages:

- Reduces the need for immediate intervention when a tablespace runs out of space
- Ensures applications will not halt or be suspended because of failures to allocate extents

To determine whether a datafile is auto-extensible, query the DBA_DATA_FILES view and examine the AUTOEXTENSIBLE column.

You can specify automatic file extension by specifying an AUTOEXTEND ON clause when you create datafiles using the following SQL statements:

- CREATE DATABASE
- ALTER DATABASE
- CREATE TABLESPACE
- ALTER TABLESPACE

You can enable or disable automatic file extension for existing datafiles, or manually resize a datafile, using the ALTER DATABASE statement. For a bigfile tablespace, you are able to perform these operations using the ALTER TABLESPACE statement.

The following example enables automatic extension for a datafile added to the users tablespace:

```
ALTER TABLESPACE users
    ADD DATAFILE '/u02/oracle/rbdb1/users03.dbf' SIZE 10M
      AUTOEXTEND ON
      NEXT 512K
      MAXSIZE 250M;
```

The value of NEXT is the minimum size of the increments added to the file when it extends. The value of MAXSIZE is the maximum size to which the file can automatically extend.

The next example disables the automatic extension for the datafile.

```
ALTER DATABASE DATAFILE '/u02/oracle/rbdb1/users03.dbf'
    AUTOEXTEND OFF;
```

See Also: *Oracle Database SQL Reference* for more information about the SQL statements for creating or altering datafiles

Manually Resizing a Datafile

You can manually increase or decrease the size of a datafile using the ALTER DATABASE statement. This enables you to add more space to your database without adding more datafiles. This is beneficial if you are concerned about reaching the maximum number of datafiles allowed in your database.

For a bigfile tablespace you can use the ALTER TABLESPACE statement to resize a datafile. You are not allowed to add a datafile to a bigfile tablespace.

Manually reducing the sizes of datafiles enables you to reclaim unused space in the database. This is useful for correcting errors in estimates of space requirements.

In the next example, assume that the datafile `/u02/oracle/rbdb1/stuff01.dbf` has extended up to 250M. However, because its tablespace now stores smaller objects, the datafile can be reduced in size.

The following statement decreases the size of datafile `/u02/oracle/rbdb1/stuff01.dbf`:

```
ALTER DATABASE DATAFILE '/u02/oracle/rbdb1/stuff01.dbf'
   RESIZE 100M;
```

Note: It is not always possible to decrease the size of a file to a specific value. It could be that the file contains data beyond the specified decreased size, in which case the database will return an error.

Altering Datafile Availability

You can take individual datafiles or tempfiles of a tablespace offline or bring them online. Offline datafiles are unavailable to the database and cannot be accessed until they are brought back online.You also have the option of taking all datafiles or tempfiles comprising a tablespace offline or online simply by specifying the name of a tablespace.

The datafiles of a read-only tablespace can be taken offline or brought online in similar fashion, but bringing a file online does not affect the read-only status of the tablespace. You cannot write to the datafile until the tablespace is returned to the read / write state.

One example of where you might be required to alter the availability of a datafile is when the database has problems writing to a datafile and automatically takes the datafile offline. Later, after resolving the problem, you can bring the datafile back online manually.

To take a datafile offline, or bring it online, you must have the ALTER DATABASE system privilege. To take all datafiles or tempfiles offline using the ALTER TABLESPACE statement, you must have the ALTER TABLESPACE or MANAGE TABLESPACE system privilege. In an Oracle Real Application Clusters environment, the database must be open in exclusive mode.

This section describes ways to alter datafile availability, and contains the following topics:

- Bringing Datafiles Online or Taking Offline in ARCHIVELOG Mode
- Taking Datafiles Offline in NOARCHIVELOG Mode
- Altering the Availability of All Datafiles or Tempfiles in a Tablespace

---

Note: You can make all datafiles of a tablespace temporarily unavailable by taking the tablespace offline. You *must* leave these files in the tablespace to bring the tablespace back online, although you can relocate or rename them following procedures similar to those shown in "Renaming and Relocating Datafiles" on page 9-11.

For more information about taking a tablespace offline, see "Taking Tablespaces Offline" on page 8-25.

Bringing Datafiles Online or Taking Offline in ARCHIVELOG Mode

To bring an individual datafile online, issue the ALTER DATABASE statement and include the DATAFILE clause.The following statement brings the specified datafile online:

```
ALTER DATABASE DATAFILE '/u02/oracle/rbdb1/stuff01.dbf' ONLINE;
```

To take the same file offline, issue the following statement:

```
ALTER DATABASE DATAFILE '/u02/oracle/rbdb1/stuff01.dbf' OFFLINE;
```

> **Note:** To use this form of the ALTER DATABASE statement, the database must be in ARCHIVELOG mode. This requirement prevents you from accidentally losing the datafile, since taking the datafile offline while in NOARCHIVELOG mode is likely to result in losing the file.

Taking Datafiles Offline in NOARCHIVELOG Mode

If a datafile becomes corrupted or missing, you must take it offline before you can open the database. To take a datafile offline when the database is in NOARCHIVELOG mode, use the ALTER DATABASE statement with both the DATAFILE and OFFLINE FOR DROP clauses.

- The OFFLINE keyword causes the database to mark the datafile OFFLINE, whether or not it is corrupted, so that you can open the database.
- The FOR DROP keywords mark the datafile for subsequent dropping. Such a datafile can no longer be brought back online.

> **Note:** This operation does not actually drop the datafile. It remains in the data dictionary, and you must drop the it yourself using an operating system command or by issuing a DROP TABLESPACE ... INCLUDING CONTENTS AND DATAFILES statement.

The following statement takes the specified datafile offline and marks it to be dropped:

```
ALTER DATABASE DATAFILE '/u02/oracle/rbdb1/users03.dbf' OFFLINE FOR DROP;
```

Altering the Availability of All Datafiles or Tempfiles in a Tablespace

Clauses of the ALTER TABLESPACE statement allow you to change the online or offline status of all of the datafiles or tempfiles within a tablespace. Specifically, the statements that affect online / offline status are:

- ALTER TABLESPACE ... DATAFILE {ONLINE | OFFLINE}
- ALTER TABLESPACE ... TEMPFILE {ONLINE | OFFLINE}

You are required only to enter the tablespace name, not the individual datafiles or tempfiles. All of the datafiles or tempfiles are affected, but the online/offline status of the tablespace itself is not changed.

In most cases the preceding ALTER TABLESPACE statements can be issued whenever the database is mounted, even if it is not open. However, the database *must not* be open if the tablespace is the SYSTEM tablespace, an undo tablespace, or the default temporary tablespace. The ALTER DATABASE DATAFILE and ALTER DATABASE TEMPFILE statements also have ONLINE/OFFLINE clauses, however in those statements you must enter all of the filenames for the tablespace.

The syntax is different from the ALTER TABLESPACE ... ONLINE|OFFLINE statement that alters tablespace availability, because that is a different operation. The ALTER TABLESPACE statement takes datafiles offline as well as the tablespace, but it cannot be used to alter the status of a temporary tablespace or its tempfile(s).

Renaming and Relocating Datafiles

You can rename datafiles to either change their names or relocate them. Some possible procedures for doing this are described in the following sections:

- Procedures for Renaming and Relocating Datafiles in a Single Tablespace
- Procedure for Renaming and Relocating Datafiles in Multiple Tablespaces

When you rename and relocate datafiles with these procedures, only the pointers to the datafiles, as recorded in the database control file, are changed. The procedures do not physically rename any operating system files, nor do they copy files at the operating system level. Renaming and relocating datafiles involves several steps. Read the steps and examples carefully before performing these procedures.

Procedures for Renaming and Relocating Datafiles in a Single Tablespace

The section suggests some procedures for renaming and relocating datafiles that can be used for a single tablespace. You must have ALTER TABLESPACE system privileges.

> **See Also:** "Taking Tablespaces Offline" on page 8-25 for more information about taking tablespaces offline in preparation for renaming or relocating datafiles

Procedure for Renaming Datafiles in a Single Tablespace

To rename datafiles in a single tablespace, complete the following steps:

1. Take the tablespace that contains the datafiles offline. The database must be open.

   For example:

   ```
   ALTER TABLESPACE users OFFLINE NORMAL;
   ```

2. Rename the datafiles using the operating system.
3. Use the ALTER TABLESPACE statement with the RENAME DATAFILE clause to change the filenames within the database.

   For example, the following statement renames the datafiles /u02/oracle/rbdb1/user1.dbf and /u02/oracle/rbdb1/user2.dbf to/u02/oracle/rbdb1/users01.dbf and /u02/oracle/rbdb1/users02.dbf, respectively:

   ```
   ALTER TABLESPACE users
       RENAME DATAFILE '/u02/oracle/rbdb1/user1.dbf',
                       '/u02/oracle/rbdb1/user2.dbf'
                    TO '/u02/oracle/rbdb1/users01.dbf',
                       '/u02/oracle/rbdb1/users02.dbf';
   ```

   Always provide complete filenames (including their paths) to properly identify the old and new datafiles. In particular, specify the old datafile name exactly as it appears in the DBA_DATA_FILES view of the data dictionary.
4. Back up the database. After making any structural changes to a database, always perform an immediate and complete backup.

Procedure for Relocating Datafiles in a Single Tablespace

Here is a sample procedure for relocating a datafile.

Assume the following conditions:

- An open database has a tablespace named users that is made up of datafiles all located on the same disk.
- The datafiles of the users tablespace are to be relocated to different and separate disk drives.
- You are currently connected with administrator privileges to the open database.
- You have a current backup of the database.

Complete the following steps:

1. If you do not know the specific file names or sizes, you can obtain this information by issuing the following query of the data dictionary view DBA_DATA_FILES:

```
SQL> SELECT FILE_NAME, BYTES FROM DBA_DATA_FILES
  2> WHERE TABLESPACE_NAME = 'USERS';

FILE_NAME                                BYTES
---------------------------------------- ----------------
/u02/oracle/rbdb1/users01.dbf            102400000
/u02/oracle/rbdb1/users02.dbf            102400000
```

2. Take the tablespace containing the datafiles offline:

```
ALTER TABLESPACE users OFFLINE NORMAL;
```

3. Copy the datafiles to their new locations and rename them using the operating system. You can copy the files using the DBMS_FILE_TRANSFER package discussed in "Copying Files Using the Database Server" on page 9-15.

> **Note:** You can temporarily exit SQL*Plus to execute an operating system command to copy a file by using the SQL*Plus HOST command.

4. Rename the datafiles within the database.

   The datafile pointers for the files that make up the users tablespace, recorded in the control file of the associated database, must now be changed from the old names to the new names.

   Use the ALTER TABLESPACE ... RENAME DATAFILE statement.

```
ALTER TABLESPACE users
    RENAME DATAFILE '/u02/oracle/rbdb1/users01.dbf',
                    '/u02/oracle/rbdb1/users02.dbf'
                 TO '/u03/oracle/rbdb1/users01.dbf',
                    '/u04/oracle/rbdb1/users02.dbf';
```

5. Back up the database. After making any structural changes to a database, always perform an immediate and complete backup.

Procedure for Renaming and Relocating Datafiles in Multiple Tablespaces

You can rename and relocate datafiles in one or more tablespaces using the ALTER DATABASE RENAME FILE statement. This method is the only choice if you want to rename or relocate datafiles of several tablespaces in one operation. You must have the ALTER DATABASE system privilege.

> **Note:** To rename or relocate datafiles of the SYSTEM tablespace, the default temporary tablespace, or the active undo tablespace you must use this ALTER DATABASE method because you cannot take these tablespaces offline.

To rename datafiles in multiple tablespaces, follow these steps.

1. Ensure that the database is mounted but closed.

   > **Note:** Optionally, the database does not have to be closed, but the datafiles (or tempfiles) must be offline.

2. Copy the datafiles to be renamed to their new locations and new names, using the operating system. You can copy the files using the DBMS_FILE_TRANSFER package discussed in "Copying Files Using the Database Server" on page 9-15.

3. Use ALTER DATABASE to rename the file pointers in the database control file.

   For example, the following statement renames the datafiles `/u02/oracle/rbdb1/sort01.dbf` and `/u02/oracle/rbdb1/user3.dbf` to `/u02/oracle/rbdb1/temp01.dbf` and `/u02/oracle/rbdb1/users03.dbf`, respectively:

   ```
   ALTER DATABASE
       RENAME FILE '/u02/oracle/rbdb1/sort01.dbf',
                   '/u02/oracle/rbdb1/user3.dbf'
                TO '/u02/oracle/rbdb1/temp01.dbf',
                   '/u02/oracle/rbdb1/users03.dbf';
   ```

   Always provide complete filenames (including their paths) to properly identify the old and new datafiles. In particular, specify the old datafile names exactly as they appear in the DBA_DATA_FILES view.

4. Back up the database. After making any structural changes to a database, always perform an immediate and complete backup.

Dropping Datafiles

There is no SQL statement that specifically drops a datafile. The only means of dropping a datafile is to drop the tablespace that contains the datafile. For example, if you want to remove a datafile from a tablespace, you could do the following:

1. Create a new tablespace
2. Move the data from the old tablespace to the new one
3. Drop the old tablespace

You can, however, drop a tempfile using the ALTER DATABASE statement. For example:

```
ALTER DATABASE TEMPFILE '/u02/oracle/data/lmtemp02.dbf' DROP
     INCLUDING DATAFILES;
```

See Also: Dropping Tablespaces on page 8-33

Verifying Data Blocks in Datafiles

If you want to configure the database to use checksums to verify data blocks, set the initialization parameter DB_BLOCK_CHECKSUM to TRUE. This causes the DBW*n* process and the direct loader to calculate a checksum for each block and to store the checksum in the block header when writing the block to disk.

The checksum is verified when the block is read, but only if DB_BLOCK_CHECKSUM is TRUE and the last write of the block stored a checksum. If corruption is detected, the database returns message ORA-01578 and writes information about the corruption to the alert file.

The default value of DB_BLOCK_CHECKSUM is TRUE. The value of this parameter can be changed dynamically using the ALTER SYSTEM statement. Regardless of the setting of this parameter, checksums are always used to verify data blocks in the SYSTEM tablespace.

See Also: *Oracle Database Reference* for more information about the DB_BLOCK_CHECKSUM initialization parameter

Copying Files Using the Database Server

You do not necessarily have to use the operating system to copy a file within a database, or transfer a file between databases as you would do when using the transportable tablespace feature. You can use the DBMS_FILE_TRANSFER package,

or you can use Streams propagation. Using Streams is not discussed in this book, but an example of using the DBMS_FILE_TRANSFER package is shown in "Copying a File on a Local File System" on page 9-16.

On UNIX systems, the owner of a file created by the DBMS_FILE_TRANSFER package is the owner of the shadow process running the instance. Normally, this owner is ORACLE. A file created using DBMS_FILE_TRANSFER is always writable and readable by all processes in the database, but non privileged users who need to read or write such a file directly may need access from a system administrator.

If the source file is an operating system file, then the destination file must also be an operating system file. Similarly, if the source file is an Automatic Storage Management file, then the destination file must also be an Automatic Storage Management file.

This section contains the following topics:

- Copying a File on a Local File System
- Third-Party File Transfer
- File Transfer and the DBMS_SCHEDULER Package
- Advanced File Transfer Mechanisms

See Also:

- *Oracle Streams Concepts and Administration*
- "Transporting Tablespaces Between Databases" on page 8-40

Copying a File on a Local File System

This section includes an example that uses the COPY_FILE procedure in the DBMS_FILE_TRANSFER package to copy a file on a local file system. The following example copies a binary file named db1.dat from the /usr/admin/source directory to the /usr/admin/destination directory as db1_copy.dat on a local file system:

1. In SQL*Plus, connect as an administrative user who can grant privileges and create directory objects using SQL.
2. Use the SQL command CREATE DIRECTORY to create a directory object for the directory from which you want to copy the file. A directory object is similar to an alias for the directory. For example, to create a directory object called SOURCE_DIR for the /usr/admin/source directory on your computer system, execute the following statement:

```
CREATE DIRECTORY SOURCE_DIR AS '/usr/admin/source';
```

3. Use the SQL command CREATE DIRECTORY to create a directory object for the directory into which you want to copy the binary file. For example, to create a directory object called DEST_DIR for the /usr/admin/destination directory on your computer system, execute the following statement:

```
CREATE DIRECTORY DEST_DIR AS '/usr/admin/destination';
```

4. Grant the required privileges to the user who will run the COPY_FILE procedure. In this example, the strmadmin user runs the procedure.

```
GRANT EXECUTE ON DBMS_FILE_TRANSFER TO strmadmin;

GRANT READ ON DIRECTORY source_dir TO strmadmin;

GRANT WRITE ON DIRECTORY dest_dir TO strmadmin;
```

5. Connect as strmadmin user:

```
CONNECT strmadmin/strmadminpw
```

6. Run the COPY_FILE procedure to copy the file:

```
BEGIN
  DBMS_FILE_TRANSFER.COPY_FILE(
        source_directory_object       =>  'SOURCE_DIR',
        source_file_name              =>  'db1.dat',
        destination_directory_object  =>  'DEST_DIR',
        destination_file_name         =>  'db1_copy.dat');
END;
/
```

Third-Party File Transfer

Although the procedures in the DBMS_FILE_TRANSFER package typically are invoked as local procedure calls, they can also be invoked as remote procedure calls. A remote procedure call lets you copy a file within a database even when you are connected to a different database. For example, you can make a copy of a file on database DB, even if you are connected to another database, by executing the following remote procedure call:

```
DBMS_FILE_TRANSFER.COPY_FILE@DB(...)
```

Using remote procedure calls enables you to copy a file between two databases, even if you are not connected to either database. For example, you can connect to database A and then transfer a file from database B to database C. In this example, database A is the third party because it is neither the source of nor the destination for the transferred file.

A third-party file transfer can both push and pull a file. Continuing with the previous example, you can perform a third-party file transfer if you have a database link from A to either B or C, and that database has a database link to the other database. Database A does not need a database link to both B and C.

For example, if you have a database link from A to B, and another database link from B to C, then you can run the following procedure at A to transfer a file from B to C:

```
DBMS_FILE_TRANSFER.PUT_FILE@B(...)
```

This configuration pushes the file.

Alternatively, if you have a database link from A to C, and another database link from C to B, then you can run the following procedure at database A to transfer a file from B to C:

```
DBMS_FILE_TRANSFER.GET_FILE@C(...)
```

This configuration pulls the file.

File Transfer and the DBMS_SCHEDULER Package

You can use the DBMS_SCHEDULER package to transfer files automatically within a single database and between databases. Third-party file transfers are also supported by the DBMS_SCHEDULER package. You can monitor a long-running file transfer done by the Scheduler using the V$SESSION_LONGOPS dynamic performance view at the databases reading or writing the file. Any database links used by a Scheduler job must be fixed user database links.

You can use a restartable Scheduler job to improve the reliability of file transfers automatically, especially if there are intermittent failures. If a file transfer fails before the destination file is closed, then you can restart the file transfer from the beginning once the database has removed any partially written destination file. Hence you should consider using a restartable Scheduler job to transfer a file if the rest of the job is restartable.

Note: If a single restartable job transfers several files, then you should consider restart scenarios in which some of the files have been transferred already and some have not been transferred yet.

Advanced File Transfer Mechanisms

You can create more sophisticated file transfer mechanisms using both the DBMS_FILE_TRANSFER package and the DBMS_SCHEDULER package. For example, when several databases have a copy of the file you want to transfer, you can consider factors such as source availability, source load, and communication bandwidth to the destination database when deciding which source database to contact first and which source databases to try if failures occur. In this case, the information about these factors must be available to you, and you must create the mechanism that considers these factors.

As another example, when early completion time is more important than load, you can submit a number of Scheduler jobs to transfer files in parallel. As a final example, knowing something about file layout on the source and destination databases enables you to minimize disk contention by performing or scheduling simultaneous transfers only if they use different I/O devices.

Mapping Files to Physical Devices

In an environment where datafiles are simply file system files or are created directly on a raw device, it is relatively straight forward to see the association between a tablespace and the underlying device. Oracle Database provides views, such as DBA_TABLESPACES, DBA_DATA_FILES, and V$DATAFILE, that provide a mapping of files onto devices. These mappings, along with device statistics can be used to evaluate I/O performance.

However, with the introduction of host based Logical Volume Managers (LVM), and sophisticated storage subsystems that provide RAID (Redundant Array of Inexpensive Disks) features, it is not easy to determine file to device mapping. This poses a problem because it becomes difficult to determine your "hottest" files when they are hidden behind a "black box". This section presents the Oracle Database approach to resolving this problem.

The following topics are contained in this section:

- Overview of Oracle Database File Mapping Interface
- How the Oracle Database File Mapping Interface Works

- Using the Oracle Database File Mapping Interface
- File Mapping Examples

Note: This section presents an overview of the Oracle Database file mapping interface and explains how to use the DBMS_STORAGE_MAP package and dynamic performance views to expose the mapping of files onto physical devices. You can more easily access this functionality through the Oracle Enterprise Manager (EM). It provides an easy to use graphical interface for mapping files to physical devices.

See the *Oracle Enterprise Manager Concepts* for more information.

Overview of Oracle Database File Mapping Interface

To acquire an understanding of I/O performance, one must have detailed knowledge of the storage hierarchy in which files reside. Oracle Database provides a mechanism to show a complete mapping of a file to intermediate layers of logical volumes to actual physical devices. This is accomplished though a set of dynamic performance views (V$ views). Using these views, you can locate the exact disk on which any block of a file resides.

To build these views, storage vendors must provide mapping libraries that are responsible for mapping their particular I/O stack elements. The database communicates with these libraries through an external non-Oracle Database process that is spawned by a background process called FMON. FMON is responsible for managing the mapping information. Oracle provides a PL/SQL package, DBMS_STORAGE_MAP, that you use to invoke mapping operations that populate the mapping views.

How the Oracle Database File Mapping Interface Works

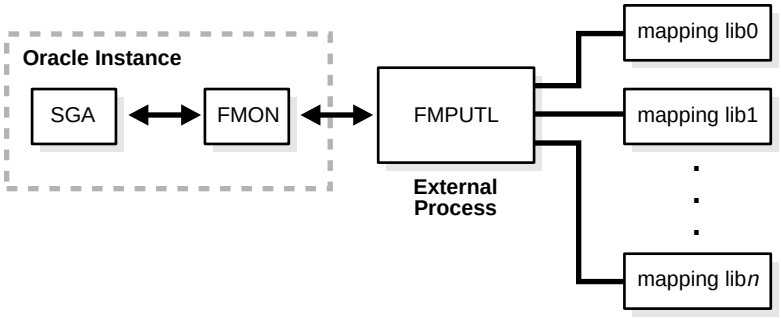
This section describes the components of the Oracle Database file mapping interface and how the interface works. It contains the following topics:

- Components of File Mapping
- Mapping Structures
- Example of Mapping Structures
- Configuration ID

Components of File Mapping

The following figure shows the components of the file mapping mechanism.

Figure 9–1 Components of File Mapping



The following sections briefly describes these components and how they work together to populate the mapping views:

- FMON
- External Process (FMPUTL)
- Mapping Libraries

FMON FMON is a background process started by the database whenever the `FILE_MAPPING` initialization parameter is set to `TRUE`. FMON is responsible for:

- Building mapping information, which is stored in the SGA. This information is composed of the following structures:
  - Files
  - File system extents
  - Elements
  - Subelements

  These structures are explained in "Mapping Structures" on page 9-22.
- Refreshing mapping information when a change occurs because of:
  - Changes to datafiles (size)
  - Addition or deletion of datafiles

  - Changes to the storage configuration (not frequent)
- Saving mapping information in the data dictionary to maintain a view of the information that is persistent across startup and shutdown operations
- Restoring mapping information into the SGA at instance startup. This avoids the need for a potentially expensive complete rebuild of the mapping information on every instance startup.

You help control this mapping using procedures that are invoked with the `DBMS_STORAGE_MAP` package.

External Process (FMPUTL) FMON spawns an external non-Oracle Database process called `FMPUTL`, that communicates directly with the vendor supplied mapping libraries. This process obtains the mapping information through all levels of the I/O stack, assuming that mapping libraries exist for all levels. On some platforms the external process requires that the `SETUID` bit is set to `ON` because root privileges are needed to map through all levels of the I/O mapping stack.

The external process is responsible for discovering the mapping libraries and dynamically loading them into its address space.

Mapping Libraries Oracle Database uses mapping libraries to discover mapping information for the elements that are owned by a particular mapping library. Through these mapping libraries information about individual I/O stack elements is communicated. This information is used to populate dynamic performance views that can be queried by users.

Mapping libraries need to exist for all levels of the stack for the mapping to be complete, and different libraries may own their own parts of the I/O mapping stack. For example, a VERITAS VxVM library would own the stack elements related to the VERITAS Volume Manager, and an EMC library would own all EMC storage specific layers of the I/O mapping stack.

Mapping libraries are vendor supplied. However, Oracle currently supplies a mapping library for EMC storage. The mapping libraries available to a database server are identified in a special file named `filemap.ora`.

Mapping Structures

The mapping structures and the Oracle Database representation of these structures are described in this section. You will need to understand this information in order to interpret the information in the mapping views.

The following are the primary structures that compose the mapping information:

- Files

  A file mapping structure provides a set of attributes for a file, including file size, number of file system extents that the file is composed of, and the file type.

- File system extents

  A file system extent mapping structure describes a contiguous chunk of blocks residing on one element. This includes the device offset, the extent size, the file offset, the type (data or parity), and the name of the element where the extent resides.

  Note: File system extents are not the same as Oracle Database extents. File system extents are physical contiguous blocks of data written to a device as managed by the file system. Oracle Database extents are logical structures managed by the database, such as tablespace extents.

- Elements

  An element mapping structure is the abstract mapping structure that describes a storage component within the I/O stack. Elements may be mirrors, stripes, partitions, RAID5, concatenated elements, and disks. These structures are the mapping building blocks.

- Subelements

  A subelement mapping structure describes the link between an element and the next elements in the I/O mapping stack. This structure contains the subelement number, size, the element name where the subelement exists, and the element offset.

All of these mapping structures are illustrated in the following example.

Example of Mapping Structures

Consider an Oracle Database which is composed of two data files X and Y. Both files X and Y reside on a file system mounted on volume A. File X is composed of two extents while file Y is composed of only one extent.

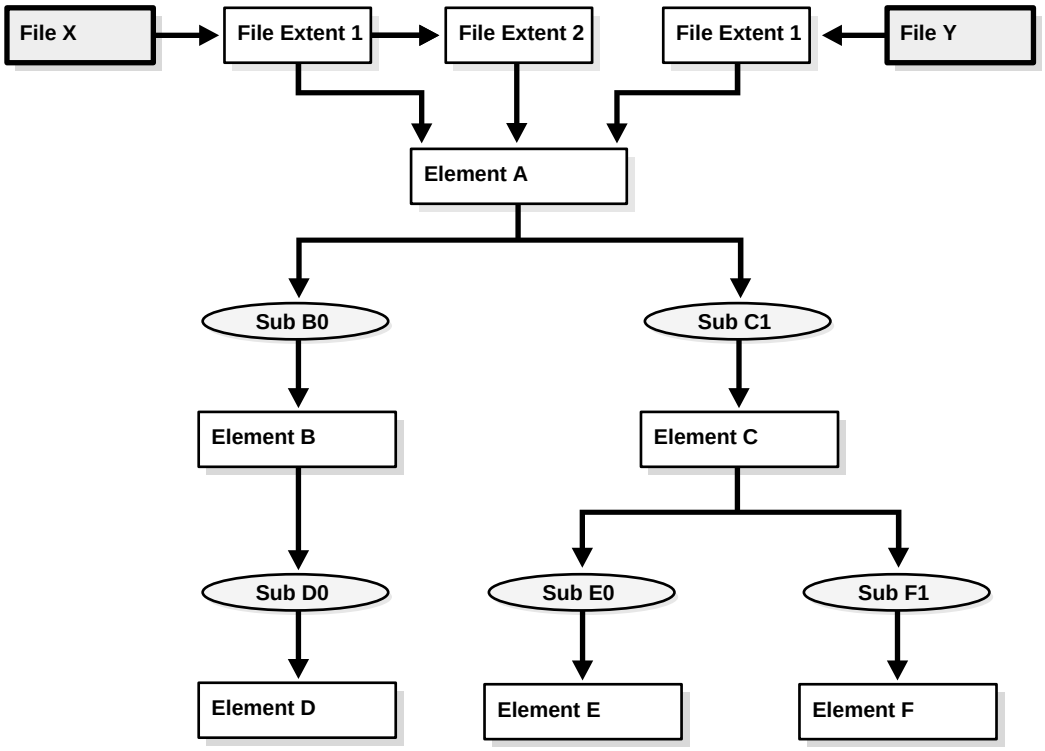
The two extents of File X and the one extent of File Y both map to Element A. Element A is striped to Elements B and C. Element A maps to Elements B and C by way of Subelements B0 and C1, respectively.

Element B is a partition of Element D (a physical disk), and is mapped to Element D by way of subelement D0.

Element C is mirrored over Elements E and F (both physical disks), and is mirrored to those physical disks by way of Subelements E0 and F1, respectively.

All of the mapping structures are illustrated in Figure 9–2.

Figure 9–2 Illustration of Mapping Structures



Note that the mapping structures represented are sufficient to describe the entire mapping information for the Oracle Database instance and consequently to map every logical block within the file into a (element name, element offset) tuple (or more in case of mirroring) at each level within the I/O stack.

Configuration ID

The configuration ID captures the version information associated with elements or files. The vendor library provides the configuration ID and updates it whenever a change occurs. Without a configuration ID, there is no way for the database to tell whether the mapping has changed.

There are two kinds of configuration IDs:

- Persistent

  These configuration IDs are persistent across instance shutdown

- nonpersistent

  The configuration IDs are not persistent across instance shutdown. The database is only capable of refreshing the mapping information while the instance is up.

Using the Oracle Database File Mapping Interface

This section discusses how to use the Oracle Database file mapping interface. It contains the following topics:

- Enabling File Mapping
- Using the DBMS_STORAGE_MAP Package
- Obtaining Information from the File Mapping Views

Enabling File Mapping

The following steps enable the file mapping feature:

1. Ensure that a valid `filemap.ora` file exists in the `$ORACLE_HOME/rdbms/filemap/etc` directory.

   Caution: While the format and content of the `filemap.ora` file is discussed here, it is for informational reasons only. The `filemap.ora` file is created by the database when your system is installed. Until such time that vendors supply there own libraries, there will be only one entry in the `filemap.ora` file, and that is the Oracle-supplied EMC library. This file should be modified manually by uncommenting this entry *only* if an EMC Symmetrix array is available.

The `filemap.ora` file is the configuration file that describes all of the available mapping libraries. FMON requires that a `filemap.ora` file exists and that it points to a valid path to mapping libraries. Otherwise, it will not start successfully.

The following row needs to be included in `filemap.ora` for each library :

`lib=`*`vendor_name:mapping_library_path`*

where:

- *vendor_name* should be `Oracle` for the EMC Symmetric library
- *mapping_library_path* is the full path of the mapping library

Note that the ordering of the libraries in this file is extremely important. The libraries are queried based on their order in the configuration file.

The file mapping service can be even started even if no mapping libraries are available. The `filemap.ora` file still needs to be present even though it is empty. In this case, the mapping service is constrained in the sense that new mapping information cannot be discovered. Only restore and drop operations are allowed in such a configuration.

2. Set the `FILE_MAPPING` initialization parameter to `TRUE`.

   ```
   FILE_MAPPING=TRUE
   ```

   The instance does not have to be shut down to set this parameter. It can be set using an `ALTER SYSTEM` statement.

3. Invoke the appropriate `DBMS_STORAGE_MAP` mapping procedure. You have two options:

   - In a cold startup scenario, the Oracle Database is just started and no mapping operation has been invoked yet. You execute the `DBMS_STORAGE_MAP.MAP_ALL` procedure to build the mapping information for the entire I/O subsystem associated with the database.
   - In a warm start scenario where the mapping information is already built, you have the option to invoke the `DBMS_STORAGE_MAP.MAP_SAVE` procedure to save the mapping information in the data dictionary. (Note that this procedure is invoked in `DBMS_STORAGE_MAP.MAP_ALL()` by default.) This forces all of the mapping information in the SGA to be flushed to disk.

Once you restart the database, use DBMS_STORAGE_MAP.RESTORE() to restore the mapping information into the SGA. If needed, DBMS_STORAGE_MAP.MAP_ALL() can be called to refresh the mapping information.

Using the DBMS_STORAGE_MAP Package

The DBMS_STORAGE_MAP package enables you to control the mapping operations. The various procedures available to you are described in the following table.

Procedure	Use to:
MAP_OBJECT	Build the mapping information for the database object identified by object name, owner, and type
MAP_ELEMENT	Build mapping information for the specified element
MAP_FILE	Build mapping information for the specified filename
MAP_ALL	Build entire mapping information for all types of database files (excluding archive logs)
DROP_ELEMENT	Drop the mapping information for a specified element
DROP_FILE	Drop the file mapping information for the specified filename
DROP_ALL	Drop all mapping information in the SGA for this instance
SAVE	Save into the data dictionary the required information needed to regenerate the entire mapping
RESTORE	Load the entire mapping information from the data dictionary into the shared memory of the instance
LOCK_MAP	Lock the mapping information in the SGA for this instance
UNLOCK_MAP	Unlock the mapping information in the SGA for this instance

See Also:

- *PL/SQL Packages and Types Reference* for a description of the DBMS_STORAGE_MAP package
- "File Mapping Examples" on page 9-29 for an example of using the DBMS_STORAGE_MAP package

Obtaining Information from the File Mapping Views

Mapping information generated by DBMS_STORAGE_MAP package is captured in dynamic performance views. Brief descriptions of these views are presented here.

View	Description
V$MAP_LIBRARY	Contains a list of all mapping libraries that have been dynamically loaded by the external process
V$MAP_FILE	Contains a list of all file mapping structures in the shared memory of the instance
V$MAP_FILE_EXTENT	Contains a list of all file system extent mapping structures in the shared memory of the instance
V$MAP_ELEMENT	Contains a list of all element mapping structures in the SGA of the instance
V$MAP_EXT_ELEMENT	Contains supplementary information for all element mapping
V$MAP_SUBELEMENT	Contains a list of all subelement mapping structures in the shared memory of the instance
V$MAP_COMP_LIST	Contains supplementary information for all element mapping structures.
V$MAP_FILE_IO_STACK	The hierarchical arrangement of storage containers for the file displayed as a series of rows. Each row represents a level in the hierarchy.

See Also: *Oracle Database Reference* for a complete description of the dynamic performance views

However, the information generated by the DBMS_STORAGE_MAP.MAP_OBJECT procedure is captured in a global temporary table named MAP_OBJECT. This table displays the hierarchical arrangement of storage containers for objects. Each row in the table represents a level in the hierarchy. A description of the MAP_OBJECT table follows.

Column	Datatype	Description
OBJECT_NAME	VARCHAR2(2000)	Name of the object
OBJECT_OWNER	VARCHAR2(2000)	Owner of the object
OBJECT_TYPE	VARCHAR2(2000)	Object type
FILE_MAP_IDX	NUMBER	File index (corresponds to FILE_MAP_IDX in V$MAP_FILE)
DEPTH	NUMBER	Element depth within the I/O stack
ELEM_IDX	NUMBER	Index corresponding to element

Column	Datatype	Description
CU_SIZE	NUMBER	Contiguous set of logical blocks of the file, in HKB (half KB) units, that is resident contiguously on the element
STRIDE	NUMBER	Number of HKB between contiguous units (CU) in the file that are contiguous on this element. Used in RAID5 and striped files.
NUM_CU	NUMBER	Number of contiguous units that are adjacent to each other on this element that are separated by STRIDE HKB in the file. In RAID5, the number of contiguous units also include the parity stripes.
ELEM_OFFSET	NUMBER	Element offset in HKB units
FILE_OFFSET	NUMBER	Offset in HKB units from the start of the file to the first byte of the contiguous units
DATA_TYPE	VARCHAR2(2000)	Datatype (DATA, PARITY, or DATA AND PARITY)
PARITY_POS	NUMBER	Position of the parity. Only for RAID5. This field is needed to distinguish the parity from the data part.
PARITY_PERIOD	NUMBER	Parity period. Only for RAID5.

File Mapping Examples

The following examples illustrates some of the powerful capabilities of the Oracle Database file mapping feature. This includes :

- The ability to map all the database files that span a particular device
- The ability to map a particular file into its corresponding devices
- The ability to map a particular database object, including its block distribution at all levels within the I/O stack

Consider an Oracle Database instance which is composed of two datafiles:

- `t_db1.f`
- `t_db2.f`

These files are created on a Solaris UFS file system mounted on a VERITAS VxVM host based striped volume, `/dev/vx/dsk/ipfdg/ipf-vol1`, that consists of the following host devices as externalized from an EMC Symmetrix array:

- `/dev/vx/rdmp/c2t1d0s2`

- /dev/vx/rdmp/c2t1d1s2

Note that the following examples require the execution of a MAP_ALL() operation.

Example 1: Map All Database Files that Span a Device

The following query returns all Oracle Database files associated with the /dev/vx/rdmp/c2t1d1s2 host device:

```
SELECT UNIQUE me.ELEM_NAME, mf.FILE_NAME
   FROM V$MAP_FILE_IO_STACK fs, V$MAP_FILE mf, V$MAP_ELEMENT me
   WHERE mf.FILE_MAP_IDX = fs.FILE_MAP_IDX
   AND me.ELEM_IDX = fs.ELEM_IDX
   AND me.ELEM_NAME = '/dev/vx/rdmp/c2t1d1s2';
```

The query results are:

```
ELEM_NAME                 FILE_NAME
------------------------  --------------------------------
/dev/vx/rdmp/c2t1d1s2     /oracle/dbs/t_db1.f
/dev/vx/rdmp/c2t1d1s2     /oracle/dbs/t_db2.f
```

Example 2: Map a File into Its Corresponding Devices

The following query displays a topological graph of the /oracle/dbs/t_db1.f datafile:

```
WITH fv AS
  (SELECT FILE_MAP_IDX, FILE_NAME FROM V$MAP_FILE
   WHERE FILE_NAME = '/oracle/dbs/t_db1.f')
SELECT fv.FILE_NAME, LPAD(' ', 4 * (LEVEL - 1)) || el.ELEM_NAME ELEM_NAME
   FROM V$MAP_SUBELEMENT sb, V$MAP_ELEMENT el, fv,
     (SELECT UNIQUE ELEM_IDX FROM V$MAP_FILE_IO_STACK io, fv
      WHERE io.FILE_MAP_IDX = fv.FILE_MAP_IDX) fs
   WHERE el.ELEM_IDX = sb.CHILD_IDX
   AND fs.ELEM_IDX = el.ELEM_IDX
   START WITH sb.PARENT_IDX IN
     (SELECT DISTINCT ELEM_IDX
      FROM V$MAP_FILE_EXTENT fe, fv
      WHERE fv.FILE_MAP_IDX = fe.FILE_MAP_IDX)
   CONNECT BY PRIOR sb.CHILD_IDX = sb.PARENT_IDX;
```

The resulting topological graph is:

```
FILE_NAME                   ELEM_NAME
--------------------------  ------------------------------------------------
/oracle/dbs/t_db1.f         _sym_plex_/dev/vx/rdsk/ipfdg/ipf-vol1_-1_-1
```

```
/oracle/dbs/t_db1.f             _sym_subdisk_/dev/vx/rdsk/ipfdg/ipf-vol1_0_0_0
/oracle/dbs/t_db1.f                 /dev/vx/rdmp/c2t1d0s2
/oracle/dbs/t_db1.f                     _sym_symdev_000183600407_00C
/oracle/dbs/t_db1.f                         _sym_hyper_000183600407_00C_0
/oracle/dbs/t_db1.f                         _sym_hyper_000183600407_00C_1
/oracle/dbs/t_db1.f             _sym_subdisk_/dev/vx/rdsk/ipfdg/ipf-vol1_0_1_0
/oracle/dbs/t_db1.f                 /dev/vx/rdmp/c2t1d1s2
/oracle/dbs/t_db1.f                     _sym_symdev_000183600407_00D
/oracle/dbs/t_db1.f                         _sym_hyper_000183600407_00D_0
/oracle/dbs/t_db1.f                         _sym_hyper_000183600407_00D_1
```

Example 3: Map a Database Object

This example displays the block distribution at all levels within the I/O stack for the `scott.bonus` table.

A `MAP_OBJECT()` operation must first be executed as follows:

```
EXECUTE DBMS_STORAGE_MAP.MAP_OBJECT('BONUS','SCOTT','TABLE');
```

The query is as follows:

```
SELECT io.OBJECT_NAME o_name, io.OBJECT_OWNER o_owner, io.OBJECT_TYPE o_type,
       mf.FILE_NAME, me.ELEM_NAME, io.DEPTH,
      (SUM(io.CU_SIZE * (io.NUM_CU - DECODE(io.PARITY_PERIOD, 0, 0,
                         TRUNC(io.NUM_CU / io.PARITY_PERIOD)))) / 2) o_size
   FROM MAP_OBJECT io, V$MAP_ELEMENT me, V$MAP_FILE mf
   WHERE io.OBJECT_NAME =  'BONUS'
   AND   io.OBJECT_OWNER = 'SCOTT'
   AND   io.OBJECT_TYPE =  'TABLE'
   AND   me.ELEM_IDX = io.ELEM_IDX
   AND   mf.FILE_MAP_IDX = io.FILE_MAP_IDX
   GROUP BY io.ELEM_IDX, io.FILE_MAP_IDX, me.ELEM_NAME, mf.FILE_NAME, io.DEPTH,
         io.OBJECT_NAME, io.OBJECT_OWNER, io.OBJECT_TYPE
   ORDER BY io.DEPTH;
```

The following is the result of the query. Note that the `o_size` column is expressed in KB.

```
O_NAME O_OWNER O_TYPE  FILE_NAME            ELEM_NAME                      DEPTH   O_SIZE
------ ------- ------  -------------------  -----------------------------  ------  ------
BONUS  SCOTT   TABLE   /oracle/dbs/t_db1.f  /dev/vx/dsk/ipfdg/ipf-vol1          0      20
BONUS  SCOTT   TABLE   /oracle/dbs/t_db1.f  _sym_plex_/dev/vx/rdsk/ipf          1      20
                                            pdg/if-vol1_-1_-1
BONUS  SCOTT   TABLE   /oracle/dbs/t_db1.f  _sym_subdisk_/dev/vx/rdsk/          2      12
                                            ipfdg/ipf-vol1_0_1_0
```

```
BONUS  SCOTT   TABLE   /oracle/dbs/t_db1.f  _sym_subdisk_/dev/vx/rdsk/ipf       2       8
                                            dg/ipf-vol1_0_2_0
BONUS  SCOTT   TABLE   /oracle/dbs/t_db1.f  /dev/vx/rdmp/c2t1d1s2               3      12
BONUS  SCOTT   TABLE   /oracle/dbs/t_db1.f  /dev/vx/rdmp/c2t1d2s2               3       8
BONUS  SCOTT   TABLE   /oracle/dbs/t_db1.f  _sym_symdev_000183600407_00D        4      12
BONUS  SCOTT   TABLE   /oracle/dbs/t_db1.f  _sym_symdev_000183600407_00E        4       8
BONUS  SCOTT   TABLE   /oracle/dbs/t_db1.f  _sym_hyper_000183600407_00D_0       5      12
BONUS  SCOTT   TABLE   /oracle/dbs/t_db1.f  _sym_hyper_000183600407_00D_1       5      12
BONUS  SCOTT   TABLE   /oracle/dbs/t_db1.f  _sym_hyper_000183600407_00E_0       6       8
BONUS  SCOTT   TABLE   /oracle/dbs/t_db1.f  _sym_hyper_000183600407_00E_1       6       8
```

Viewing Datafile Information

The following data dictionary views provide useful information about the datafiles of a database:

View	Description
DBA_DATA_FILES	Provides descriptive information about each datafile, including the tablespace to which it belongs and the file ID. The file ID can be used to join with other views for detail information.
DBA_EXTENTS USER_EXTENTS	DBA view describes the extents comprising all segments in the database. Contains the file ID of the datafile containing the extent. USER view describes extents of the segments belonging to objects owned by the current user.
DBA_FREE_SPACE USER_FREE_SPACE	DBA view lists the free extents in all tablespaces. Includes the file ID of the datafile containing the extent. USER view lists the free extents in the tablespaces accessible to the current user.
V$DATAFILE	Contains datafile information from the control file
V$DATAFILE_HEADER	Contains information from datafile headers

This example illustrates the use of one of these views, V$DATAFILE.

```
SELECT NAME,
    FILE#,
    STATUS,
    CHECKPOINT_CHANGE# "CHECKPOINT"
  FROM   V$DATAFILE;

NAME                                 FILE#     STATUS       CHECKPOINT
-------------------------------      -----     -------       ----------
```

```
/u01/oracle/rbdb1/system01.dbf          1     SYSTEM              3839
/u02/oracle/rbdb1/temp01.dbf            2     ONLINE              3782
/u02/oracle/rbdb1/users03.dbf           3     OFFLINE             3782
```

FILE# lists the file number of each datafile; the first datafile in the SYSTEM tablespace created with the database is always file 1. STATUS lists other information about a datafile. If a datafile is part of the SYSTEM tablespace, its status is SYSTEM (unless it requires recovery). If a datafile in a non-SYSTEM tablespace is online, its status is ONLINE. If a datafile in a non-SYSTEM tablespace is offline, its status can be either OFFLINE or RECOVER. CHECKPOINT lists the final SCN (system change number) written for the most recent checkpoint of a datafile.

> **See Also:** *Oracle Database Reference* for complete descriptions of these views

10

Managing the Undo Tablespace

This chapter describes how to manage the undo tablespace, which stores information used to roll back changes to the Oracle Database. It contains the following topics:

- What Is Undo?
- Introduction to Automatic Undo Management
- Sizing the Undo Tablespace
- Managing Undo Tablespaces
- Monitoring the Undo Tablespace
- Flashback Features and Undo Space
- Migration to Automatic Undo Management
- Best Practices

> **See Also:** Part III, "Automated File and Storage Management" for information about creating an undo tablespace whose datafiles are both created and managed by the Oracle Database server.

What Is Undo?

Every Oracle Database must have a method of maintaining information that is used to roll back, or undo, changes to the database. Such information consists of records of the actions of transactions, primarily before they are committed. These records are collectively referred to as **undo**.

Undo records are used to:

- Roll back transactions when a ROLLBACK statement is issued

- Recover the database
- Provide read consistency
- Analyze data as of an earlier point in time by using Flashback Query
- Recover from logical corruptions using Flashback features

When a ROLLBACK statement is issued, undo records are used to undo changes that were made to the database by the uncommitted transaction. During database recovery, undo records are used to undo any uncommitted changes applied from the redo log to the datafiles. Undo records provide read consistency by maintaining the before image of the data for users who are accessing the data at the same time that another user is changing it.

Earlier releases of Oracle Database used rollback segments to store undo. Oracle9*i* introduced automatic undo management, which simplifies undo space management by eliminating the complexities associated with rollback segment management. Oracle strongly recommends that you use undo tablespace to manage undo rather than rollback segments.

> **See Also:** "Migration to Automatic Undo Management" on page 10-17 for information on how to migrate to automatic undo management

Introduction to Automatic Undo Management

This section introduces the concepts of Automatic Undo Management and discusses the following topics:

- Overview of Automatic Undo Management
- Undo Retention
- Retention Guarantee

Overview of Automatic Undo Management

In past releases, when you used the rollback segment method of managing undo space, you were said to be operating in the manual undo management mode. Now, you use the undo tablespace method, and you are said to be operating in the automatic undo management mode. You determine the mode at instance startup using the UNDO_MANAGEMENT initialization parameter. The default value for this parameter is MANUAL. You set it to AUTO to enable automatic undo management.

Notes:

- You cannot use both methods in the same database instance, although for migration purposes it is possible, for example, to create undo tablespaces in a database that is using rollback segments, or to drop rollback segments in a database that is using undo tablespaces. However, you must shut down and restart your database in order to effect the switch to another method of managing undo.
- When operating in automatic undo management mode, any manual undo management SQL statements are ignored and no error message is issued. For example, ALTER ROLLBACK SEGMENT statements will be ignored.

The following initialization parameter setting causes the STARTUP command to start an instance in automatic undo management mode:

```
UNDO_MANAGEMENT = AUTO
```

An undo tablespace must be available, into which the database will store undo records. The default undo tablespace is created at database creation, or an undo tablespace can be created explicitly. The methods of creating an undo tablespace are explained in "Creating an Undo Tablespace" on page 10-9

When the instance starts up, the database automatically selects for use the first available undo tablespace. If there is no undo tablespace available, the instance starts, but uses the SYSTEM rollback segment for undo. This is not recommended in normal circumstances, and an alert message is written to the alert file to warn that the system is running without an undo tablespace. ORA-01552 errors are issued for any attempts to write non-SYSTEM related undo to the SYSTEM rollback segment.

If the database contains multiple undo tablespaces, you can optionally specify at startup that you want an Oracle Database instance to use a specific undo tablespace. This is done by setting the UNDO_TABLESPACE initialization parameter. For example:

```
UNDO_TABLESPACE = undotbs_01
```

In this case, if you have not already created the undo tablespace (in this example, undotbs_01), the STARTUP command will fail. The UNDO_TABLESPACE parameter can be used to assign a specific undo tablespace to an instance in an Oracle Real Application Clusters environment.

The following is a summary of the initialization parameters for automatic undo management mode:

Initialization Parameter	Description
UNDO_MANAGEMENT	If AUTO, use automatic undo management mode. If MANUAL, use manual undo management mode. The default is MANUAL.
UNDO_TABLESPACE	An optional dynamic parameter specifying the name of an undo tablespace to use. This parameter should be used only when the database has multiple undo tablespaces and you want to direct the database instance to use a particular undo tablespace.
UNDO_RETENTION	A dynamic parameter specifying the minimum length of time to retain undo. The default is 900 seconds. The setting of this parameter should take into account any flashback requirements of the system.

If the initialization parameter file contains parameters relating to manual undo management, they are ignored.

> **See Also:** *Oracle Database Reference* for complete descriptions of initialization parameters used in automatic undo management mode

Undo Retention

Committed undo information normally is lost when its undo space is overwritten by a newer transaction. However, for consistent read purposes, long-running queries sometimes require old undo information for undoing changes and producing older images of data blocks. The success of several Flashback features can also depend upon older undo information.

Automatic Tuning of Undo Retention

Oracle Database 10*g* automatically tunes undo retention by collecting database use statistics and estimating undo capacity needs for the successful completion of the queries. You can set a low threshold value for the UNDO_RETENTION parameter so that the system retains the undo for at least the time specified in the parameter, provided that the current undo tablespace has enough space. Under space constraint conditions, the system may retain undo for a shorter duration than that specified by the low threshold value in order to allow DML operations to succeed.

In order to guarantee the success of queries even at the price of compromising the success of DML operations, you can enable retention guarantee. The RETENTION GUARANTEE clause of the CREATE UNDO TABLESPACE and CREATE DATABASE statements ensures that undo information is not overwritten. This option must be used with caution, because it can cause DML operations to fail if the undo tablespace is not big enough. However, with proper settings, long-running queries can complete without risk of receiving the "snapshot too old" message, and you can guarantee a time window in which the execution of Flashback features will succeed.

Setting the UNDO_RETENTION Initialization Parameter

The default value for the UNDO_RETENTION parameter is 900. Retention is specified in units of seconds. This parameter determines the low threshold value of undo retention. The system retains undo for at least the time specified in this parameter. The setting of this parameter should account for any flashback requirements of the system.

You can set the UNDO_RETENTION parameter initially in the initialization parameter file that is used by the STARTUP process:

```
UNDO_RETENTION = 1800
```

You can change the UNDO_RETENTION parameter value at any time using the ALTER SYSTEM statement:

```
ALTER SYSTEM SET UNDO_RETENTION = 2400
```

The effect of the UNDO_RETENTION parameter is immediate, but it can only be honored if the current undo tablespace has enough space. If an active transaction requires undo space and the undo tablespace does not have available space, then the system starts reusing unexpired undo space. This action can potentially cause some queries to fail with the "snapshot too old" message.

The amount of time for which undo is retained for Oracle Database for the current undo tablespace can be obtained by querying the TUNED_UNDORETENTION column of the V$UNDOSTAT dynamic performance view.

Automatic tuning of undo retention is not supported for LOBs. The RETENTION value for LOB columns is set to the value of the UNDO_RETENTION parameter.

Retention Guarantee

Oracle Database 10*g* lets you guarantee undo retention. When you enable this option, the database never overwrites unexpired undo data--that is, undo data

whose age is less than the undo retention period. This option is disabled by default, which means that the database can overwrite the unexpired undo data in order to avoid failure of DML operations if there is not enough free space left in the undo tablespace.

By enabling the guarantee option, you instruct the database not to overwrite unexpired undo data even if it means risking failure of currently active DML operations. Therefore, use caution when using this feature. A typical use of the guarantee option is when you want to ensure deterministic and predictable behavior of Flashback Query by guaranteeing the availability of the required undo data.

You enable the guarantee option by specifying the `RETENTION GUARANTEE` clause for the undo tablespace when it is created by either the `CREATE DATABASE` or `CREATE UNDO TABLESPACE` statement. Or, you can later specify this clause in an `ALTER TABLESPACE` statement. You *do not* guarantee that unexpired undo is preserved if you specify the `RETENTION NOGUARANTEE` clause.

You can use the `DBA_TABLESPACES` view to determine the `RETENTION` setting for the undo tablespace. A column named `RETENTION` will contain a value on `GUARANTEE`, `NOGUARANTEE`, or `NOT APPLY` (used for tablespaces other than the undo tablespace).

Sizing the Undo Tablespace

You can size the undo tablespace appropriately either by using automatic extension of the undo tablespace or by manually estimating the space you will need for undo. This section discusses both methods.

Using Auto-Extensible Tablespaces

Oracle Database supports automatic extension of the undo tablespace to facilitate capacity planning of the undo tablespace in the production environment. When the system is first running in the production environment, you may be unsure of the space requirements of the undo tablespace. In this case, you can enable automatic extension for datafiles of the undo tablespace so that they automatically increase in size when more space is needed. By combining automatic extension of the undo tablespace with automatically tuned undo retention, you can ensure that long-running queries will succeed by guaranteeing the undo required for such queries.

After the system has stabilized and you are more familiar with undo space requirements, Oracle recommends that you set the maximum size of the tablespace to be slightly (10%) more than the current size of the undo tablespace.

Sizing Fixed-Size Undo Tablespaces

If you have decided on a fixed-size undo tablespace, the Undo Advisor can help you estimate needed capacity, and you can then calculate the amount of retention your system will need. You can access the Undo Advisor through Enterprise Manager or through the DBMS_ADVISOR PL/SQL package. Enterprise Manager is the preferred method of accessing the advisor. For more information on using the Undo Advisor through EM, please refer to *Oracle 2 Day DBA*.

The Undo Advisor relies for its analysis on data collected in the Automatic Workload Repository (AWR). An adjustment to the collection interval and retention period for AWR statistics can affect the precision and the type of recommendations the advisor produces. Please refer to "Automatic Workload Repository" on page 1-26 for additional information.

The Undo Advisor PL/SQL Interface

Oracle Database provides an Undo Advisor that provides advice on and helps automate the establishment of your undo environment. You activate the Undo Advisor by creating an undo advisor task through the advisor framework. The following example creates an undo advisor task to evaluate the undo tablespace. The name of the advisor is 'Undo Advisor'. The analysis is based on Automatic Workload Repository snapshots, which you must specify by setting parameters START_SNAPSHOT and END_SNAPSHOT. In the following example, the START_SNAPSHOT is "1" and END_SNAPSHOT is "2".

```
DECLARE
   tid    NUMBER;
   tname  VARCHAR2(30);
   oid    NUMBER;
   BEGIN
   DBMS_ADVISOR.CREATE_TASK('Undo Advisor', tid, tname, 'Undo Advisor Task');
   DBMS_ADVISOR.CREATE_OBJECT(tname, 'UNDO_TBS', null, null, null, 'null', oid);
   DBMS_ADVISOR.SET_TASK_PARAMETER(tname, 'TARGET_OBJECTS', oid);
   DBMS_ADVISOR.SET_TASK_PARAMETER(tname, 'START_SNAPSHOT', 1);
   DBMS_ADVISOR.SET_TASK_PARAMETER(tname, 'END_SNAPSHOT', 2);
   DBMS_ADVISOR.execute_task(tname);
   end;
/
```

Once you have created the advisor task, you can view the output and recommendations in the Automatic Database Diagnostic Monitor in Enterprise Manager. This information is also available in the DBA_ADVISOR_* data dictionary views.

See Also:

- *Oracle 2 Day DBA* for more information on using advisors and "Segment Advisor" on page 13-29 for an example of creating an advisor task for a different advisor
- *Oracle Database Reference* for information about the DBA_ADVISOR_* data dictionary views

Calculating the Space Requirements For Undo Retention

You can calculate space requirements manually using the following formula:

```
UndoSpace = UR * UPS + overhead
```

where:

- UndoSpace is the number of undo blocks
- UR is UNDO_RETENTION in seconds. This value should take into consideration long-running queries and any flashback requirements.
- UPS is undo blocks for each second
- overhead is the small overhead for metadata (transaction tables, bitmaps, and so forth)

As an example, if UNDO_RETENTION is set to 2 hours, and the transaction rate (UPS) is 200 undo blocks for each second, with a 4K block size, the required undo space is computed as follows:

(2 * 3600 * 200 * 4K) = 5.8GBs

Such computation can be performed by using information in the V$UNDOSTAT view. In the steady state, you can query the view to obtain the transaction rate. The overhead figure can also be obtained from the view.

Managing Undo Tablespaces

This section describes the various steps involved in undo tablespace management and contains the following sections:

Creating an Undo Tablespace

There are two methods of creating an undo tablespace. The first method creates the undo tablespace when the CREATE DATABASE statement is issued. This occurs when you are creating a new database, and the instance is started in automatic undo management mode (UNDO_MANAGEMENT = AUTO). The second method is used with an existing database. It uses the CREATE UNDO TABLESPACE statement.

You cannot create database objects in an undo tablespace. It is reserved for system-managed undo data.

Oracle Database enables you to create a single-file undo tablespace. Single-file, or bigfile, tablespaces are discussed in "Bigfile Tablespaces" on page 8-9.

Using CREATE DATABASE to Create an Undo Tablespace

You can create a specific undo tablespace using the UNDO TABLESPACE clause of the CREATE DATABASE statement.

The following statement illustrates using the UNDO TABLESPACE clause in a CREATE DATABASE statement. The undo tablespace is named `undotbs_01` and one datafile, `/u01/oracle/rbdb1/undo0101.dbf`, is allocated for it.

```
CREATE DATABASE rbdb1
     CONTROLFILE REUSE
     .
     .
     .
     UNDO TABLESPACE undotbs_01 DATAFILE '/u01/oracle/rbdb1/undo0101.dbf';
```

If the undo tablespace cannot be created successfully during CREATE DATABASE, the entire CREATE DATABASE operation fails. You must clean up the database files, correct the error and retry the CREATE DATABASE operation.

The CREATE DATABASE statement also lets you create a single-file undo tablespace at database creation. This is discussed in "Supporting Bigfile Tablespaces During Database Creation" on page 2-23.

> **See Also:** *Oracle Database SQL Reference* for the syntax for using the CREATE DATABASE statement to create an undo tablespace

Using the CREATE UNDO TABLESPACE Statement

The CREATE UNDO TABLESPACE statement is the same as the CREATE TABLESPACE statement, but the UNDO keyword is specified. The database determines most of the attributes of the undo tablespace, but you can specify the DATAFILE clause.

This example creates the undotbs_02 undo tablespace:

```
CREATE UNDO TABLESPACE undotbs_02
     DATAFILE '/u01/oracle/rbdb1/undo0201.dbf' SIZE 2M REUSE AUTOEXTEND ON;
```

You can create more than one undo tablespace, but only one of them can be active at any one time.

> **See Also:** *Oracle Database SQL Reference* for the syntax for using the CREATE UNDO TABLESPACE statement to create an undo tablespace

Altering an Undo Tablespace

Undo tablespaces are altered using the ALTER TABLESPACE statement. However, since most aspects of undo tablespaces are system managed, you need only be concerned with the following actions:

- Adding a datafile
- Renaming a datafile
- Bringing a datafile online or taking it offline
- Beginning or ending an open backup on a datafile
- Enabling and disabling undo retention guarantee

These are also the only attributes you are permitted to alter.

If an undo tablespace runs out of space, or you want to prevent it from doing so, you can add more files to it or resize existing datafiles.

The following example adds another datafile to undo tablespace undotbs_01:

```
ALTER TABLESPACE undotbs_01
     ADD DATAFILE '/u01/oracle/rbdb1/undo0102.dbf' AUTOEXTEND ON NEXT 1M
         MAXSIZE UNLIMITED;
```

You can use the ALTER DATABASE ... DATAFILE statement to resize or extend a datafile.

See Also:

- "Changing Datafile Size" on page 9-6
- *Oracle Database SQL Reference* for ALTER TABLESPACE syntax

Dropping an Undo Tablespace

Use the DROP TABLESPACE statement to drop an undo tablespace. The following example drops the undo tablespace undotbs_01:

```
DROP TABLESPACE undotbs_01;
```

An undo tablespace can only be dropped if it is not currently used by any instance. If the undo tablespace contains any outstanding transactions (for example, a transaction died but has not yet been recovered), the DROP TABLESPACE statement fails. However, since DROP TABLESPACE drops an undo tablespace even if it contains unexpired undo information (within retention period), you must be careful not to drop an undo tablespace if undo information is needed by some existing queries.

DROP TABLESPACE for undo tablespaces behaves like DROP TABLESPACE ... INCLUDING CONTENTS. All contents of the undo tablespace are removed.

See Also: *Oracle Database SQL Reference* for DROP TABLESPACE syntax

Switching Undo Tablespaces

You can switch from using one undo tablespace to another. Because the UNDO_TABLESPACE initialization parameter is a dynamic parameter, the ALTER SYSTEM SET statement can be used to assign a new undo tablespace.

The following statement switches to a new undo tablespace:

```
ALTER SYSTEM SET UNDO_TABLESPACE = undotbs_02;
```

Assuming undotbs_01 is the current undo tablespace, after this command successfully executes, the instance uses undotbs_02 in place of undotbs_01 as its undo tablespace.

If any of the following conditions exist for the tablespace being switched to, an error is reported and no switching occurs:

- The tablespace does not exist
- The tablespace is not an undo tablespace
- The tablespace is already being used by another instance (in a RAC environment only)

The database is online while the switch operation is performed, and user transactions can be executed while this command is being executed. When the switch operation completes successfully, all transactions started after the switch operation began are assigned to transaction tables in the new undo tablespace.

The switch operation does not wait for transactions in the old undo tablespace to commit. If there are any pending transactions in the old undo tablespace, the old undo tablespace enters into a `PENDING OFFLINE` mode (status). In this mode, existing transactions can continue to execute, but undo records for new user transactions cannot be stored in this undo tablespace.

An undo tablespace can exist in this `PENDING OFFLINE` mode, even after the switch operation completes successfully. A `PENDING OFFLINE` undo tablespace cannot be used by another instance, nor can it be dropped. Eventually, after all active transactions have committed, the undo tablespace automatically goes from the `PENDING OFFLINE` mode to the `OFFLINE` mode. From then on, the undo tablespace is available for other instances (in an Oracle Real Application Cluster environment).

If the parameter value for `UNDO TABLESPACE` is set to '' (two single quotes), then the current undo tablespace is switched out and the next available undo tablespace is switched in. Use this statement with care, because if there is no undo tablespace available, the `SYSTEM` rollback segment is used. This causes ORA-01552 errors to be issued for any attempts to write non-`SYSTEM` related undo to the `SYSTEM` rollback segment.

The following example unassigns the current undo tablespace:

```
ALTER SYSTEM SET UNDO_TABLESPACE = '';
```

Establishing User Quotas for Undo Space

The Oracle Database Resource Manager can be used to establish user quotas for undo space. The Database Resource Manager directive `UNDO_POOL` allows DBAs to limit the amount of undo space consumed by a group of users (resource consumer group).

You can specify an undo pool for each consumer group. An undo pool controls the amount of total undo that can be generated by a consumer group. When the total undo generated by a consumer group exceeds its undo limit, the current UPDATE transaction generating the redo is terminated. No other members of the consumer group can perform further updates until undo space is freed from the pool.

When no UNDO_POOL directive is explicitly defined, users are allowed unlimited undo space.

See Also: Chapter 24, "Using the Database Resource Manager"

Monitoring the Undo Tablespace

This section lists views that are useful for viewing information about undo space in the automatic undo management mode and provides some examples. In addition to views listed here, you can obtain information from the views available for viewing tablespace and datafile information. Please refer to "Viewing Datafile Information" on page 9-32 for information on getting information about those views.

Oracle Database also provides proactive help in managing tablespace disk space use by alerting you when tablespaces run low on available space. Please refer to "Managing Space in Tablespaces" on page 13-9 for information on how to set alert thresholds for the undo tablespace.

In addition to the proactive undo space alerts, Oracle Database also provides alerts if your system has long-running queries that cause SNAPSHOT TOO OLD errors. To prevent excessive alerts, the long query alert is issued at most once every 24 hours. When the alert is generated, you can check the Undo Advisor Page of Enterprise Manager to get more information about the undo tablespace.

The following dynamic performance views are useful for obtaining space information about the undo tablespace:

View	Description
V$UNDOSTAT	Contains statistics for monitoring and tuning undo space. Use this view to help estimate the amount of undo space required for the current workload. The database also uses this information to help tune undo usage in the system. This view is meaningful only in automatic undo management mode.
V$ROLLSTAT	For automatic undo management mode, information reflects behavior of the undo segments in the undo tablespace

View	Description
V$TRANSACTION	Contains undo segment information
DBA_UNDO_EXTENTS	Shows the status and size of each extent in the undo tablespace.
WRH$_UNDOSTAT	Contains statistical snapshots of V$UNDOSTAT information. Please refer to *Oracle 2 Day DBA* for more information.
WRH$_ROLLSTAT	Contains statistical snapshots of V$ROLLSTAT information. Please refer to *Oracle 2 Day DBA* for more information.

See Also: *Oracle Database Reference* for complete descriptions of the views used in automatic undo management mode

The V$UNDOSTAT view is useful for monitoring the effects of transaction execution on undo space in the current instance. Statistics are available for undo space consumption, transaction concurrency, the tuning of undo retention, and the length and SQL ID of long-running queries in the instance.

Each row in the view contains statistics collected in the instance for a ten-minute interval. The rows are in descending order by the BEGIN_TIME column value. Each row belongs to the time interval marked by (BEGIN_TIME, END_TIME). Each column represents the data collected for the particular statistic in that time interval. The first row of the view contains statistics for the (partial) current time period. The view contains a total of 1008 rows, spanning a 7 day cycle.

The following example shows the results of a query on the V$UNDOSTAT view.

```
SELECT TO_CHAR(BEGIN_TIME, 'MM/DD/YYYY HH24:MI:SS') BEGIN_TIME,
       TO_CHAR(END_TIME, 'MM/DD/YYYY HH24:MI:SS') END_TIME,
       UNDOTSN, UNDOBLKS, TXNCOUNT, MAXCONCURRENCY AS "MAXCON"
       FROM v$UNDOSTAT WHERE rownum <= 144;

BEGIN_TIME          END_TIME              UNDOTSN   UNDOBLKS   TXNCOUNT     MAXCON
------------------- ------------------- ---------- ---------- ---------- ----------
10/28/2003 14:25:12 10/28/2003 14:32:17          8         74   12071108          3
10/28/2003 14:15:12 10/28/2003 14:25:12          8         49   12070698          2
10/28/2003 14:05:12 10/28/2003 14:15:12          8        125   12070220          1
10/28/2003 13:55:12 10/28/2003 14:05:12          8         99   12066511          3
...
10/27/2003 14:45:12 10/27/2003 14:55:12          8         15   11831676          1
10/27/2003 14:35:12 10/27/2003 14:45:12          8        154   11831165          2

144 rows selected.
```

The preceding example shows how undo space is consumed in the system for the previous 24 hours from the time 14:35:12 on 10/27/2003.

Flashback Features and Undo Space

Your Oracle Database includes several features that are based upon undo information and that allow administrators and users to access database information from a previous point in time. These features are part of the overall flashback strategy incorporated into the database and include:

- Flashback Query
- Flashback Versions Query
- Flashback Transaction Query
- Flashback Table

The retention period for undo information is an important factor for the successful execution of Flashback features. It determines how far back in time a database version can be established. Specifically, you must choose an undo retention interval that is long enough to enable users to construct a snapshot of the database for the oldest version of the database that they are interested in. For example, if an application requires that a version of the database be available reflecting its content 12 hours previously, then `UNDO_RETENTION` must be set to 43200.

You might also want to guarantee that unexpired undo is *not* overwritten by specifying the `RETENTION GUARANTEE` clause for the undo tablespace as described in "Undo Retention" on page 10-4.

Flashback Query

Using Oracle Flashback Query feature, users or applications can execute queries as of a previous time in the database. Application developers can use Flashback Query to design an application that allows users to correct their mistakes with minimal DBA intervention. You, as the DBA, need only configure the undo tablespace with an appropriate size and undo retention period. No further action on your part should be required.

The Oracle-supplied `DBMS_FLASHBACK` package implements Flashback Query at the session level. At the object level, Flashback Query uses the `AS OF` clause of the `SELECT` statement to specify the previous point in time for which you wish to view data.

See Also:

- *Oracle Database Application Developer's Guide - Fundamentals* for information about using the Flashback Query feature
- *PL/SQL Packages and Types Reference* for a description of the DBMS_FLASHBACK package
- *Oracle Database SQL Reference* for a description of the AS OF clause of the SELECT statement

Flashback Versions Query

The Flashback Version Query feature enables users to query the history of a given row. A SELECT statement that specifies a VERSIONS clause returns individual versions of a row between two specified SCNs (system change numbers) or timestamps. The UNDO_RETENTION initialization parameter must be set to a value that is large enough to cover the period of time specified in the VERSIONS clause. Otherwise, not all rows can be retrieved.

The VERSIONS clause can also be used in subqueries of DML and DDL statements.

See Also:

- *Oracle Database Application Developer's Guide - Fundamentals* for information about using the Flashback Version Query feature
- *Oracle Database SQL Reference* for a description of the VERSIONS clause of the SELECT statement

Flashback Transaction Query

Oracle Database provides a view, FLASHBACK_TRANSACTION_QUERY, that enables you to identify changes made by a particular transaction, or by all transactions issued within a specified period of time. One use for this view could be if a user finds, by using the Flashback Transaction Query feature, that a row value has been changed inappropriately. Querying the FLASHBACK_TRANSACTION_QUERY view can provide specific details of the transaction or transactions that changed the row value.

See Also: "Auditing Table Changes Using Flashback Transaction Query" on page 14-33 for more information about using the Flashback Transaction Query feature

Flashback Table

The FLASHBACK TABLE statement lets users recover a table to a previous point in time. It provides a fast, online solution for recovering a table that has been accidentally modified or deleted by a user or application. The UNDO_RETENTION initialization parameter must be set to a value that is large enough to cover the period specified in the FLASHBACK TABLE statement.

> **See Also:** "Recovering Tables Using the Flashback Table Feature" on page 14-34 for more information about the FLASHBACK TABLE statement

Migration to Automatic Undo Management

If you are still using rollback segments to manage undo space, Oracle strongly recommends that you migrate your database to automatic undo management. Oracle Database provides a function that provides information on how to size your new undo tablespace based on the configuration and usage of the rollback segments in your system. DBA privileges are required to execute this function:

```
DECLARE
   utbsiz_in_MB NUMBER;
BEGIN
   utbsiz_in_MB ;= DBMS_UNDO_ADV.RBU_MIGRATION;
end;
/
```

The function returns the sizing information directly.

Best Practices

This following list of recommendations will help you manage your undo space to best advantage.

- You need not set a value for the UNDO_RETENTION parameter unless your system has flashback or LOB retention requirements.
- Allow 10 to 20% extra space in your undo tablespace to provide for some fluctuation in your workload.
- Set the warning and critical alert thresholds for the undo tablespace alert properly. Please refer to "Managing Space in Tablespaces" on page 13-9 for information on how to set alert thresholds for the undo tablespace.

- To tune SQL queries or to check on runaway queries, use the value of the SQLID column provided in the long query or in the V$UNDOSTAT or WRH$_UNDOSTAT views to retrieve SQL text and other details on the SQL from V$SQL view.
- Always use indexes for Flashback Version Query.

Part III

Automated File and Storage Management

Part III describes how to use the Oracle-managed files and Automatic Storage Management features to simplify management of your database files.

You use the Oracle-managed files feature to specify file system directories in which Oracle automatically creates and manages files for you at the database object level. For example, you need only specify that you want to create a tablespace, you do not need to specify the `DATAFILE` clause. This feature works well with a logical volume manager (LVM).

Automatic Storage Management provides a logical volume manager that is integrated into the Oracle database and eliminates the need for you to purchase a third party product. Oracle creates Oracle-managed files within "disk groups" that you specify and provides redundancy and striping.

This part contains the following chapters:

11

Using Oracle-Managed Files

This chapter discusses the use of the Oracle-managed files and contains the following topics:

What Are Oracle-Managed Files?

Using Oracle-managed files simplifies the administration of an Oracle Database. Oracle-managed files eliminate the need for you, the DBA, to directly manage the operating system files comprising an Oracle Database. You specify operations in terms of database objects rather than filenames. The database internally uses standard file system interfaces to create and delete files as needed for the following database structures:

- Tablespaces
- Redo log files
- Control files
- Archived logs
- Block change tracking files
- Flashback logs
- RMAN backups

Through initialization parameters, you specify the file system directory to be used for a particular type of file. The database then ensures that a unique file, an Oracle-managed file, is created and deleted when no longer needed.

This feature does not affect the creation or naming of administrative files such as trace files, audit files, alert files, and core files.

> **See Also:** Chapter 12, "Using Automatic Storage Management" for information about the Oracle Database integrated storage management system that extends the power of Oracle-managed files. With Oracle-managed files, files are created and managed automatically for you, but with Automatic Storage Management you get the additional benefits of features such as file redundancy and striping, without the need to purchase a third-party logical volume manager.

Who Can Use Oracle-Managed Files?

Oracle-managed files are most useful for the following types of databases:

- Databases that are supported by the following:
  - A logical volume manager that supports striping/RAID and dynamically extensible logical volumes
  - A file system that provides large, extensible files
- Low end or test databases

The Oracle-managed files feature is not intended to ease administration of systems that use raw disks. This feature provides better integration with operating system functionality for disk space allocation. Since there is no operating system support for allocation of raw disks (it is done manually), this feature cannot help. On the other hand, because Oracle-managed files require that you use the operating system file system (unlike raw disks), you lose control over how files are laid out on the disks and thus, you lose some I/O tuning ability.

What Is a Logical Volume Manager?

A logical volume manager (LVM) is a software package available with most operating systems. Sometimes it is called a logical disk manager (LDM). It allows pieces of multiple physical disks to be combined into a single contiguous address space that appears as one disk to higher layers of software. An LVM can make the logical volume have better capacity, performance, reliability, and availability

characteristics than any of the underlying physical disks. It uses techniques such as mirroring, striping, concatenation, and RAID 5 to implement these characteristics.

Some LVMs allow the characteristics of a logical volume to be changed after it is created, even while it is in use. The volume may be resized or mirrored, or it may be relocated to different physical disks.

What Is a File System?

A file system is a data structure built inside a contiguous disk address space. A file manager (FM) is a software package that manipulates file systems, but it is sometimes called the file system. All operating systems have file managers. The primary task of a file manager is to allocate and deallocate disk space into files within a file system.

A file system allows the disk space to be allocated to a large number of files. Each file is made to appear as a contiguous address space to applications such as Oracle Database. The files may not actually be contiguous within the disk space of the file system. Files can be created, read, written, resized, and deleted. Each file has a name associated with it that is used to refer to the file.

A file system is commonly built on top of a logical volume constructed by an LVM. Thus all the files in a particular file system have the same performance, reliability, and availability characteristics inherited from the underlying logical volume. A file system is a single pool of storage that is shared by all the files in the file system. If a file system is out of space, then none of the files in that file system can grow. Space available in one file system does not affect space in another file system. However some LVM/FM combinations allow space to be added or removed from a file system.

An operating system can support multiple file systems. Multiple file systems are constructed to give different storage characteristics to different files as well as to divide the available disk space into pools that do not affect each other.

Benefits of Using Oracle-Managed Files

Consider the following benefits of using Oracle-managed files:

- They make the administration of the database easier.

  There is no need to invent filenames and define specific storage requirements. A consistent set of rules is used to name all relevant files. The file system defines the characteristics of the storage and the pool where it is allocated.

- They reduce corruption caused by administrators specifying the wrong file.

Each Oracle-managed file and filename is unique. Using the same file in two different databases is a common mistake that can cause very large down times and loss of committed transactions. Using two different names that refer to the same file is another mistake that causes major corruptions.

- They reduce wasted disk space consumed by obsolete files.

  Oracle Database automatically removes old Oracle-managed files when they are no longer needed. Much disk space is wasted in large systems simply because no one is sure if a particular file is still required. This also simplifies the administrative task of removing files that are no longer required on disk and prevents the mistake of deleting the wrong file.

- They simplify creation of test and development databases.

  You can minimize the time spent making decisions regarding file structure and naming, and you have fewer file management tasks. You can focus better on meeting the actual requirements of your test or development database.

- Oracle-managed files make development of portable third-party tools easier.

  Oracle-managed files eliminate the need to put operating system specific file names in SQL scripts.

Oracle-Managed Files and Existing Functionality

Using Oracle-managed files does not eliminate any existing functionality. Existing databases are able to operate as they always have. New files can be created as managed files while old ones are administered in the old way. Thus, a database can have a mixture of Oracle-managed and unmanaged files.

Enabling the Creation and Use of Oracle-Managed Files

The following initialization parameters allow the database server to use the Oracle-managed files feature:

Initialization Parameter	Description
DB_CREATE_FILE_DEST	Defines the location of the default file system directory where the database creates datafiles or tempfiles when no file specification is given in the creation operation. Also used as the default file system directory for redo log and control files if DB_CREATE_ONLINE_LOG_DEST_n is not specified.

Initialization Parameter	Description
DB_CREATE_ONLINE_LOG_DEST_n	Defines the location of the default file system directory for redo log files and control file creation when no file specification is given in the creation operation. You can use this initialization parameter multiple times, where *n* specifies a multiplexed copy of the redo log or control file. You can specify up to five multiplexed copies.
DB_RECOVERY_FILE_DEST	Defines the location of the default file system directory where the database creates RMAN backups when no format option is used, archived logs when no other local destination is configured, and flashback logs. Also used as the default file system directory for redo log and control files if DB_CREATE_ONLINE_LOG_DEST_n is not specified.

The file system directory specified by either of these parameters must already exist: the database does not create it. The directory must also have permissions to allow the database to create the files in it.

The default location is used whenever a location is not explicitly specified for the operation creating the file. The database creates the filename, and a file thus created is an Oracle-managed file.

Both of these initialization parameters are dynamic, and can be set using the ALTER SYSTEM or ALTER SESSION statement.

See Also:

- *Oracle Database Reference* for additional information about initialization parameters
- "How Oracle-Managed Files Are Named" on page 11-8

Setting the DB_CREATE_FILE_DEST Initialization Parameter

Include the DB_CREATE_FILE_DEST initialization parameter in your initialization parameter file to identify the default location for the database server to create:

- Datafiles
- Tempfiles
- Redo log files
- Control files

- Block change tracking files

You specify the name of a file system directory that becomes the default location for the creation of the operating system files for these entities. The following example sets `/u01/oradata` as the default directory to use when creating Oracle-managed files:

```
DB_CREATE_FILE_DEST = '/u01/oradata'
```

Setting the DB_RECOVERY_FILE_DEST Parameter

Include the `DB_RECOVERY_FILE_DEST` and `DB_RECOVERY_FILE_DEST_SIZE` parameters in your initialization parameter file to identify the default location in which Oracle Database should create:

- Redo log files
- Control files
- RMAN backups (datafile copies, control file copies, backup pieces, control file autobackups)
- Archived logs
- Flashback logs

You specify the name of file system directory that becomes the default location for creation of the operating system files for these entities. For example:

```
DB_RECOVERY_FILE_DEST      = '/u01/oradata'
DB_RECOVERY_FILE_DEST_SIZE = 20G
```

Setting the DB_CREATE_ONLINE_LOG_DEST_n Initialization Parameter

Include the `DB_CREATE_ONLINE_LOG_DEST_`*n* initialization parameter in your initialization parameter file to identify the default location for the database server to create:

- Redo log files
- Control files

You specify the name of a file system directory that becomes the default location for the creation of the operating system files for these entities. You can specify up to five multiplexed locations.

For the creation of redo log files and control files only, this parameter overrides any default location specified in the `DB_CREATE_FILE_DEST` and `DB_RECOVERY_`

FILE_DEST initialization parameters. If you do not specify a DB_CREATE_FILE_DEST parameter, but you do specify the DB_CREATE_ONLINE_LOG_DEST_*n* parameter, then only redo log files and control files can be created as Oracle-managed files.

It is recommended that you specify at least two parameters. For example:

```
DB_CREATE_ONLINE_LOG_DEST_1 = '/u02/oradata'
DB_CREATE_ONLINE_LOG_DEST_2 = '/u03/oradata'
```

This allows multiplexing, which provides greater fault-tolerance for the redo log and control file if one of the destinations fails.

Creating Oracle-Managed Files

If you have met any of the following conditions, then Oracle Database creates Oracle-managed files for you, as appropriate, when no file specification is given in the creation operation:

- You have included any of the DB_CREATE_FILE_DEST, DB_REDOVERY_FILE_DEST, or DB_CREATE_ONLINE_LOG_DEST_*n* initialization parameters in your initialization parameter file.
- You have issued the ALTER SYSTEM statement to dynamically set any of DB_RECOVERY_FILE_DEST, DB_CREATE_FILE_DEST, or DB_CREATE_ONLINE_LOG_DEST_*n* initialization parameters
- You have issued the ALTER SESSION statement to dynamically set any of the DB_CREATE_FILE_DEST, DB_RECOVERY_FILE_DEST, or DB_CREATE_ONLINE_LOG_DEST_*n* initialization parameters.

If a statement that creates an Oracle-managed file finds an error or does not complete due to some failure, then any Oracle-managed files created by the statement are automatically deleted as part of the recovery of the error or failure. However, because of the large number of potential errors that can occur with file systems and storage subsystems, there can be situations where you must manually remove the files using operating system commands. When an Oracle-managed file is created, its filename is written to the alert file. This information can be used to find the file if it is necessary to manually remove the file.

The following topics are discussed in this section:

- How Oracle-Managed Files Are Named
- Creating Oracle-Managed Files at Database Creation

- Creating Datafiles for Tablespaces Using Oracle-Managed Files
- Creating Tempfiles for Temporary Tablespaces Using Oracle-Managed Files
- Creating Control Files Using Oracle-Managed Files
- Creating Redo Log Files Using Oracle-Managed Files
- Creating Archived Logs Using Oracle-Managed Files

How Oracle-Managed Files Are Named

The filenames of Oracle-managed files comply with the Optimal Flexible Architecture (OFA) standard for file naming. The assigned names are intended to meet the following requirements:

- Database files are easily distinguishable from all other files.
- Files of one database type are easily distinguishable from other database types.
- Files are clearly associated with important attributes specific to the file type. For example, a datafile name may include the tablespace name to allow for easy association of datafile to tablespace, or an archived log name may include the thread, sequence, and creation date.

No two Oracle-managed files are given the same name. The name that is used for creation of an Oracle-managed file is constructed from three sources:

- The default creation location
- A file name template that is chosen based on the type of the file. The template also depends on the operating system platform and whether or not automatic storage management is used.
- A unique string created by the Oracle Database server or the operating system. This ensures that file creation does not damage an existing file and that the file cannot be mistaken for some other file.

As a specific example, filenames for Oracle-managed files have the following format on a Solaris file system:

```
<destination_prefix>/o1_mf_%t_%u_.dbf
```

where:

- <*destination_prefix*> is <*destination_location*>/<*db_unique_name*>/<*datafile*>

  where:

  - <*destination_location*> is the location specified in DB_CREATE_FILE_DEST
  - <*db_unique_name*> is the globally unique name (DB_UNIQUE_NAME initialization parameter) of the target database. If there is no DB_UNIQUE_NAME parameter, then the DB_NAME initialization parameter value is used.
- %t is the tablespace name.
- %u is an eight-character string that guarantees uniqueness

For example, assume the following parameter settings:

```
DB_CREATE_FILE_DEST   = /u01/oradata
DB_UNIQUE_NAME = PAYROLL
```

Then an example datafile name would be:

```
/u01/oradata/PAYROLL/datafile/o1_mf_tbs1_2ixh90q_.dbf
```

Names for other file types are similar. Names on other platforms are also similar, subject to the constraints of the naming rules of the platform.

The examples on the following pages use Oracle-managed file names as they might appear with a Solaris file system as an OMF destination.

Caution: Do not rename an Oracle-managed file. The database identifies an Oracle-managed file based on its name. If you rename the file, the database is no longer able to recognize it as an Oracle-managed file and will not manage the file accordingly.

Creating Oracle-Managed Files at Database Creation

The behavior of the CREATE DATABASE statement for creating database structures when using Oracle-managed files is discussed in this section.

> **See Also:** *Oracle Database SQL Reference* for a description of the CREATE DATABASE statement

Specifying Control Files at Database Creation

At database creation, the control file is created in the files specified by the CONTROL_FILES initialization parameter. If the CONTROL_FILES parameter is not set and at least one of the initialization parameters required for the creation of Oracle-managed files is set, then an Oracle-managed control file is created in the

default control file destinations. In order of precedence, the default destination is defined as follows:

- One or more control files as specified in the DB_CREATE_ONLINE_LOG_DEST_*n* initialization parameter. The file in the first directory is the primary control file. When DB_CREATE_ONLINE_LOG_DEST_*n* is specified, the database does not create a control file in DB_CREATE_FILE_DEST or in DB_RECOVERY_FILE_DEST (the flash recovery area).
- If no value is specified for DB_CREATE_ONLINE_LOG_DEST_*n*, but values are set for both the DB_CREATE_FILE_DEST and DB_RECOVERY_FILE_DEST, then the database creates one control file in each location. The location specified in DB_CREATE_FILE_DEST is the primary control file.
- If a value is specified only for DB_CREATE_FILE_DEST, then the database creates one control file in that location.
- If a value is specified only for DB_RECOVERY_FILE_DEST, then the database creates one control file in that location.

If the CONTROL_FILES parameter is not set and none of these initialization parameters are set, then the Oracle Database default behavior is operating system dependent. At least one copy of a control file is created in an operating system dependent default location. Any copies of control files created in this fashion are not Oracle-managed files, and you must add a CONTROL_FILES initialization parameter to any initialization parameter file.

If the database creates an Oracle-managed control file, and if there is a server parameter file, then the database creates a CONTROL_FILES initialization parameter entry in the server parameter file. If there is no server parameter file, then you must manually include a CONTROL_FILES initialization parameter entry in the text initialization parameter file.

See Also: Chapter 5, "Managing Control Files"

Specifying Redo Log Files at Database Creation

The LOGFILE clause is not required in the CREATE DATABASE statement, and omitting it provides a simple means of creating Oracle-managed redo log files. If the LOGFILE clause is omitted, then redo log files are created in the default redo log file destinations. In order of precedence, the default destination is defined as follows:

- If either the DB_CREATE_ONLINE_LOG_DEST_*n* is set, then the database creates a log file member in each directory specified, up to the value of the MAXLOGMEMBERS initialization parameter.

- If the DB_CREATE_ONLINE_LOG_DEST_*n* parameter is not set, but both the DB_CREATE_FILE_DEST and DB_RECOVERY_FILE_DEST initialization parameters are set, then the database creates one Oracle-managed log file member in each of those locations. The log file in the DB_CREATE_FILE_DEST destination is the first member.
- If only the DB_CREATE_FILE_DEST initialization parameter is specified, then the database creates a log file member in that location.
- If only the DB_RECOVERY_FILE_DEST initialization parameter is specified, then the database creates a log file member in that location.

The default size of an Oracle-managed redo log file is 100 MB.

Optionally, you can create Oracle-managed redo log files, and override default attributes, by including the LOGFILE clause but omitting a filename. Redo log files are created the same way, except for the following: If no filename is provided in the LOGFILE clause of CREATE DATABASE, and none of the initialization parameters required for creating Oracle-managed files are provided, then the CREATE DATABASE statement fails.

See Also: Chapter 6, "Managing the Redo Log"

Specifying the SYSTEM and SYSAUX Tablespace Datafiles at Database Creation

The DATAFILE or SYSAUX DATAFILE clause is not required in the CREATE DATABASE statement, and omitting it provides a simple means of creating Oracle-managed datafiles for the SYSTEM and SYSAUX tablespaces. If the DATAFILE clause is omitted, then one of the following actions occurs:

- If DB_CREATE_FILE_DEST is set, then one Oracle-managed datafile for the SYSTEM tablespace and another for the SYSAUX tablespace are created in the DB_CREATE_FILE_DEST directory.
- If DB_CREATE_FILE_DEST is not set, then the database creates one SYSTEM, and one SYSAUX, tablespace datafile whose name and size are operating system dependent. Any SYSTEM or SYSAUX tablespace datafile created in this manner is not an Oracle-managed file.

The default size for an Oracle-managed datafile is 100 MB and the file is autoextensible. When autoextension is required, the database extends the datafile by its existing size or 100 MB, whichever is smaller. You can also explicitly specify the autoextensible unit using the NEXT parameter of the STORAGE clause when you specify the datafile (in a CREATE or ALTER TABLESPACE operation).

Optionally, you can create an Oracle-managed datafile for the SYSTEM or SYSAUX tablespace and override default attributes. This is done by including the DATAFILE clause, omitting a filename, but specifying overriding attributes. When a filename is not supplied and the DB_CREATE_FILE_DEST parameter is set, an Oracle-managed datafile for the SYSTEM or SYSAUX tablespace is created in the DB_CREATE_FILE_DEST directory with the specified attributes being overridden. However, if a filename is not supplied and the DB_CREATE_FILE_DEST parameter is not set, then the CREATE DATABASE statement fails.

When overriding the default attributes of an Oracle-managed file, if a SIZE value is specified but no AUTOEXTEND clause is specified, then the datafile is *not* autoextensible.

Specifying the Undo Tablespace Datafile at Database Creation

The DATAFILE subclause of the UNDO TABLESPACE clause is optional and a filename is not required in the file specification. If a filename is not supplied and the DB_CREATE_FILE_DEST parameter is set, then an Oracle-managed datafile is created in the DB_CREATE_FILE_DEST directory. If DB_CREATE_FILE_DEST is not set, then the statement fails with a syntax error.

The UNDO TABLESPACE clause itself is optional in the CREATE DATABASE statement. If it is not supplied, and automatic undo management mode is enabled, then a default undo tablespace named SYS_UNDOTBS is created and a 10 MB datafile that is autoextensible is allocated as follows:

- If DB_CREATE_FILE_DEST is set, then an Oracle-managed datafile is created in the indicated directory.
- If DB_CREATE_FILE_DEST is not set, then the datafile location is operating system specific.

See Also: Chapter 10, "Managing the Undo Tablespace"

Specifying the Default Temporary Tablespace Tempfile at Database Creation

The TEMPFILE subclause is optional for the DEFAULT TEMPORARY TABLESPACE clause and a filename is not required in the file specification. If a filename is not supplied and the DB_CREATE_FILE_DEST parameter set, then an Oracle-managed tempfile is created in the DB_CREATE_FILE_DEST directory. If DB_CREATE_FILE_DEST is not set, then the CREATE DATABASE statement fails with a syntax error.

The DEFAULT TEMPORARY TABLESPACE clause itself is optional. If it is not specified, then no default temporary tablespace is created.

The default size for an Oracle-managed tempfile is 100 MB and the file is autoextensible with an unlimited maximum size.

CREATE DATABASE Statement Using Oracle-Managed Files: Examples

This section contains examples of the CREATE DATABASE statement when using the Oracle-managed files feature.

CREATE DATABASE: Example 1 This example creates a database with the following Oracle-managed files:

- A SYSTEM tablespace datafile in directory /u01/oradata that is 100 MB and autoextensible up to an unlimited size.
- A SYSAUX tablespace datafile in directory /u01/oradata that is 100 MB and autoextensible up to an unlimited size. The tablespace is locally managed with automatic segment-space management.
- Two online log groups with two members of 100 MB each, one each in /u02/oradata and /u03/oradata.
- If automatic undo management mode is enabled, then an undo tablespace datafile in directory /u01/oradata that is 10 MB and autoextensible up to an unlimited size. An undo tablespace named SYS_UNDOTBS is created.
- If no CONTROL_FILES initialization parameter is specified, then two control files, one each in /u02/oradata and /u03/oradata. The control file in /u02/oradata is the primary control file.

The following parameter settings relating to Oracle-managed files, are included in the initialization parameter file:

```
DB_CREATE_FILE_DEST = '/u01/oradata'
DB_CREATE_ONLINE_LOG_DEST_1 = '/u02/oradata'
DB_CREATE_ONLINE_LOG_DEST_2 = '/u03/oradata'
```

The following statement is issued at the SQL prompt:

```
SQL> CREATE DATABASE sample;
```

CREATE DATABASE: Example 2 This example creates a database with the following Oracle-managed files:

- A 100 MB SYSTEM tablespace datafile in directory /u01/oradata that is autoextensible up to an unlimited size.

- A SYSAUX tablespace datafile in directory /u01/oradata that is 100 MB and autoextensible up to an unlimited size. The tablespace is locally managed with automatic segment-space management.
- Two redo log files of 100 MB each in directory /u01/oradata. They are not multiplexed.
- An undo tablespace datafile in directory /u01/oradata that is 10 MB and autoextensible up to an unlimited size. An undo tablespace named SYS_UNDOTBS is created.
- A control file in /u01/oradata.

In this example, it is assumed that:

- No DB_CREATE_ONLINE_LOG_DEST_*n* initialization parameters are specified in the initialization parameter file.
- No CONTROL_FILES initialization parameter was specified in the initialization parameter file.
- Automatic undo management mode is enabled.

The following statements are issued at the SQL prompt:

```
SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST = '/u01/oradata';
SQL> CREATE DATABASE sample2;
```

This database configuration is not recommended for a production database. The example illustrates how a very low-end database or simple test database can easily be created. To better protect this database from failures, at least one more control file should be created and the redo log should be multiplexed.

CREATE DATABASE: Example 3 In this example, the file size for the Oracle-managed files for the default temporary tablespace and undo tablespace are specified. A database with the following Oracle-managed files is created:

- A 400 MB SYSTEM tablespace datafile in directory /u01/oradata. Because SIZE is specified, the file in not autoextensible.
- A 200 MB SYSAUX tablespace datafile in directory /u01/oradata. Because SIZE is specified, the file in not autoextensible. The tablespace is locally managed with automatic segment-space management.
- Two redo log groups with two members of 100 MB each, one each in directories /u02/oradata and /u03/oradata.

- For the default temporary tablespace `dflt_ts`, a 10 MB tempfile in directory `/u01/oradata`. Because `SIZE` is specified, the file in not autoextensible.
- For the undo tablespace `undo_ts`, a 10 MB datafile in directory `/u01/oradata`. Because `SIZE` is specified, the file in not autoextensible.
- If no `CONTROL_FILES` initialization parameter was specified, then two control files, one each in directories `/u02/oradata` and `/u03/oradata`. The control file in `/u02/oradata` is the primary control file.

The following parameter settings are included in the initialization parameter file:

```
DB_CREATE_FILE_DEST = '/u01/oradata'
DB_CREATE_ONLINE_LOG_DEST_1 = '/u02/oradata'
DB_CREATE_ONLINE_LOG_DEST_2 = '/u03/oradata'
```

The following statement is issued at the SQL prompt:

```
SQL> CREATE DATABASE sample3 DATAFILE SIZE 400M
2>   SYSAUX DATAFILE SIZE 200M
3>   DEFAULT TEMPORARY TABLESPACE dflt_ts TEMPFILE SIZE 10M
4>   UNDO TABLESPACE undo_ts DATAFILE SIZE 10M;
```

Creating Datafiles for Tablespaces Using Oracle-Managed Files

The following statements that can create datafiles are relevant to the discussion in this section:

- `CREATE TABLESPACE`
- `CREATE UNDO TABLESPACE`
- `ALTER TABLESPACE ... ADD DATAFILE`

When creating a tablespace, either a regular tablespace or an undo tablespace, the `DATAFILE` clause is optional. When you include the `DATAFILE` clause the filename is optional. If the `DATAFILE` clause or filename is not provided, then the following rules apply:

- If the `DB_CREATE_FILE_DEST` initialization parameter is specified, then an Oracle-managed datafile is created in the location specified by the parameter.
- If the `DB_CREATE_FILE_DEST` initialization parameter is not specified, then the statement creating the datafile fails.

When you add a datafile to a tablespace with the `ALTER TABLESPACE ... ADD DATAFILE` statement the filename is optional. If the filename is not specified, then the same rules apply as discussed in the previous paragraph.

By default, an Oracle-managed datafile for a regular tablespace is 100 MB and is autoextensible with an unlimited maximum size. However, if in your DATAFILE clause you override these defaults by specifying a SIZE value (and no AUTOEXTEND clause), then the datafile is *not* autoextensible.

See Also:

- "Specifying the SYSTEM and SYSAUX Tablespace Datafiles at Database Creation" on page 11-11
- "Specifying the Undo Tablespace Datafile at Database Creation" on page 11-12
- Chapter 8, "Managing Tablespaces"

CREATE TABLESPACE: Examples

The following are some examples of creating tablespaces with Oracle-managed files.

See Also: *Oracle Database SQL Reference* for a description of the CREATE TABLESPACE statement

CREATE TABLESPACE: Example 1 The following example sets the default location for datafile creations to /u01/oradata and then creates a tablespace tbs_1 with a datafile in that location. The datafile is 100 MB and is autoextensible with an unlimited maximum size.

```
SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST = '/u01/oradata';
SQL> CREATE TABLESPACE tbs_1;
```

CREATE TABLESPACE: Example 2 This example creates a tablespace named tbs_2 with a datafile in the directory /u01/oradata. The datafile initial size is 400 MB, and because the SIZE clause is specified, the datafile is not autoextensible.

The following parameter setting is included in the initialization parameter file:

```
DB_CREATE_FILE_DEST = '/u01/oradata'
```

The following statement is issued at the SQL prompt:

```
SQL> CREATE TABLESPACE tbs_2 DATAFILE SIZE 400M;
```

CREATE TABLESPACE: Example 3 This example creates a tablespace named tbs_3 with an autoextensible datafile in the directory /u01/oradata with a maximum size of 800 MB and an initial size of 100 MB:

The following parameter setting is included in the initialization parameter file:

```
DB_CREATE_FILE_DEST = '/u01/oradata'
```

The following statement is issued at the SQL prompt:

```
SQL> CREATE TABLESPACE tbs_3 DATAFILE AUTOEXTEND ON MAXSIZE 800M;
```

CREATE TABLESPACE: Example 4 The following example sets the default location for datafile creations to `/u01/oradata` and then creates a tablespace named `tbs_4` in that directory with two datafiles. Both datafiles have an initial size of 200 MB, and because a `SIZE` value is specified, they are not autoextensible

```
SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST = '/u01/oradata';
SQL> CREATE TABLESPACE tbs_4 DATAFILE SIZE 200M SIZE 200M;
```

CREATE UNDO TABLESPACE: Example

The following example creates an undo tablespace named `undotbs_1` with a datafile in the directory `/u01/oradata`. The datafile for the undo tablespace is 100 MB and is autoextensible with an unlimited maximum size.

The following parameter setting is included in the initialization parameter file:

```
DB_CREATE_FILE_DEST = '/u01/oradata'
```

The following statement is issued at the SQL prompt:

```
SQL> CREATE UNDO TABLESPACE undotbs_1;
```

> **See Also:** *Oracle Database SQL Reference* for a description of the `CREATE UNDO TABLESPACE` statement

ALTER TABLESPACE: Example

This example adds an Oracle-managed autoextensible datafile to the `tbs_1` tablespace. The datafile has an initial size of 100 MB and a maximum size of 800 MB.

The following parameter setting is included in the initialization parameter file:

```
DB_CREATE_FILE_DEST = '/u01/oradata'
```

The following statement is entered at the SQL prompt:

```
SQL> ALTER TABLESPACE tbs_1 ADD DATAFILE AUTOEXTEND ON MAXSIZE 800M;
```

> **See Also:** *Oracle Database SQL Reference* for a description of the ALTER TABLESPACE statement

Creating Tempfiles for Temporary Tablespaces Using Oracle-Managed Files

The following statements that create tempfiles are relevant to the discussion in this section:

- CREATE TEMPORARY TABLESPACE
- ALTER TABLESPACE ... ADD TEMPFILE

When creating a temporary tablespace the TEMPFILE clause is optional. If you include the TEMPFILE clause, then the filename is optional. If the TEMPFILE clause or filename is not provided, then the following rules apply:

- If the DB_CREATE_FILE_DEST initialization parameter is specified, then an Oracle-managed tempfile is created in the location specified by the parameter.
- If the DB_CREATE_FILE_DEST initialization parameter is not specified, then the statement creating the tempfile fails.

When you add a tempfile to a tablespace with the ALTER TABLESPACE ... ADD TEMPFILE statement the filename is optional. If the filename is not specified, then the same rules apply as discussed in the previous paragraph.

When overriding the default attributes of an Oracle-managed file, if a SIZE value is specified but no AUTOEXTEND clause is specified, then the datafile is *not* autoextensible.

> **See Also:** "Specifying the Default Temporary Tablespace Tempfile at Database Creation" on page 11-12

CREATE TEMPORARY TABLESPACE: Example

The following example sets the default location for datafile creations to /u01/oradata and then creates a tablespace named temptbs_1 with a tempfile in that location. The tempfile is 100 MB and is autoextensible with an unlimited maximum size.

```
SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST = '/u01/oradata';
SQL> CREATE TEMPORARY TABLESPACE temptbs_1;
```

> **See Also:** *Oracle Database SQL Reference* for a description of the CREATE TABLESPACE statement

ALTER TABLESPACE ... ADD TEMPFILE: Example

The following example sets the default location for datafile creations to `/u03/oradata` and then adds a tempfile in the default location to a tablespace named `temptbs_1`. The tempfile initial size is 100 MB. It is autoextensible with an unlimited maximum size.

```
SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST = '/u03/oradata';
SQL> ALTER TABLESPACE TBS_1 ADD TEMPFILE;
```

> **See Also:** *Oracle Database SQL Reference* for a description of the `ALTER TABLESPACE` statement

Creating Control Files Using Oracle-Managed Files

When you issue the `CREATE CONTROLFILE` statement, a control file is created (or reused, if `REUSE` is specified) in the files specified by the `CONTROL_FILES` initialization parameter. If the `CONTROL_FILES` parameter is not set, then the control file is created in the default control file destinations. The default destination is determined according to the precedence documented in "Specifying Control Files at Database Creation" on page 11-9.

If Oracle Database creates an Oracle-managed control file, and there is a server parameter file, then the database creates a `CONTROL_FILES` initialization parameter for the server parameter file. If there is no server parameter file, then you must create a `CONTROL_FILES` initialization parameter manually and include it in the initialization parameter file.

If the datafiles in the database are Oracle-managed files, then the database-generated filenames for the files must be supplied in the `DATAFILE` clause of the statement.

If the redo log files are Oracle-managed files, then the `NORESETLOGS` or `RESETLOGS` keyword determines what can be supplied in the `LOGFILE` clause:

- If the `NORESETLOGS` keyword is used, then the database-generated filenames for the Oracle-managed redo log files must be supplied in the `LOGFILE` clause.
- If the `RESETLOGS` keyword is used, then the redo log file names can be supplied as with the `CREATE DATABASE` statement. See "Specifying Redo Log Files at Database Creation" on page 11-10.

The sections that follow contain examples of using the `CREATE CONTROLFILE` statement with Oracle-managed files.

See Also:

- *Oracle Database SQL Reference* for a description of the CREATE CONTROLFILE statement
- "Specifying Control Files at Database Creation" on page 11-9

CREATE CONTROLFILE Using NORESETLOGS Keyword: Example

The following CREATE CONTROLFILE statement is generated by an ALTER DATABASE BACKUP CONTROLFILE TO TRACE statement for a database with Oracle-managed datafiles and redo log files:

```
CREATE CONTROLFILE
     DATABASE sample
     LOGFILE
       GROUP 1 ('/u01/oradata/SAMPLE/onlinelog/o1_mf_1_o220rtt9_.log',
                '/u02/oradata/SAMPLE/onlinelog/o1_mf_1_v2o0b2i3_.log')
                 SIZE 100M,
       GROUP 2 ('/u01/oradata/SAMPLE/onlinelog/o1_mf_2_p22056iw_.log',
                '/u02/oradata/SAMPLE/onlinelog/o1_mf_2_p02rcyg3_.log')
                 SIZE 100M
     NORESETLOGS
     DATAFILE '/u01/oradata/SAMPLE/datafile/o1_mf_system_xu34ybm2_.dbf'
              SIZE 100M,
              '/u01/oradata/SAMPLE/datafile/o1_mf_sysaux_aawbmz51_.dbf'
              SIZE 100M,
              '/u01/oradata/SAMPLE/datafile/o1_mf_sys_undotbs_apqbmz51_.dbf'
              SIZE 100M
     MAXLOGFILES 5
     MAXLOGHISTORY 100
     MAXDATAFILES 10
     MAXINSTANCES 2
     ARCHIVELOG;
```

CREATE CONTROLFILE Using RESETLOGS Keyword: Example

The following is an example of a CREATE CONTROLFILE statement with the RESETLOGS option. Some combination of DB_CREATE_FILE_DEST, DB_RECOVERY_FILE_DEST, and DB_CREATE_ONLINE_LOG_DEST_*n* or must be set.

```
CREATE CONTROLFILE
     DATABASE sample
     RESETLOGS
     DATAFILE '/u01/oradata/SAMPLE/datafile/o1_mf_system_aawbmz51_.dbf',
              '/u01/oradata/SAMPLE/datafile/o1_mf_sysaux_axybmz51_.dbf',
```

```
        '/u01/oradata/SAMPLE/datafile/o1_mf_sys_undotbs_azzbmz51_.dbf'
SIZE 100M
MAXLOGFILES 5
MAXLOGHISTORY 100
MAXDATAFILES 10
MAXINSTANCES 2
ARCHIVELOG;
```

Later, you must issue the ALTER DATABASE OPEN RESETLOGS statement to re-create the redo log files. This is discussed in "Using the ALTER DATABASE OPEN RESETLOGS Statement" on page 11-22. If the previous log files are Oracle-managed files, then they are not deleted.

Creating Redo Log Files Using Oracle-Managed Files

Redo log files are created at database creation time. They can also be created when you issue either of the following statements:

- ALTER DATABASE ADD LOGFILE
- ALTER DATABASE OPEN RESETLOGS

> **See Also:** *Oracle Database SQL Reference* for a description of the ALTER DATABASE statement

Using the ALTER DATABASE ADD LOGFILE Statement

The ALTER DATABASE ADD LOGFILE statement lets you later add a new group to your current redo log. The filename in the ADD LOGFILE clause is optional if you are using Oracle-managed files. If a filename is not provided, then a redo log file is created in the default log file destination. The default destination is determined according to the precedence documented in "Specifying Redo Log Files at Database Creation" on page 11-10.

If a filename is not provided and you have not provided one of the initialization parameters required for creating Oracle-managed files, then the statement returns an error.

The default size for an Oracle-managed log file is 100 MB.

You continue to add and drop redo log file members by specifying complete filenames.

See Also:

- "Specifying Redo Log Files at Database Creation" on page 11-10
- "Creating Control Files Using Oracle-Managed Files" on page 11-19

Adding New Redo Log Files: Example The following example creates a log group with a member in `/u01/oradata` and another member in `/u02/oradata`. The size of each log file is 100 MB.

The following parameter settings are included in the initialization parameter file:

```
DB_CREATE_ONLINE_LOG_DEST_1 = '/u01/oradata'
DB_CREATE_ONLINE_LOG_DEST_2 = '/u02/oradata'
```

The following statement is issued at the SQL prompt:

```
SQL> ALTER DATABASE ADD LOGFILE;
```

Using the ALTER DATABASE OPEN RESETLOGS Statement

If you previously created a control file specifying `RESETLOGS` and either did not specify filenames or specified nonexistent filenames, then the database creates redo log files for you when you issue the `ALTER DATABASE OPEN RESETLOGS` statement. The rules for determining the directories in which to store redo log files, when none are specified in the control file, are the same as those discussed in "Specifying Redo Log Files at Database Creation" on page 11-10.

Creating Archived Logs Using Oracle-Managed Files

Archived logs are created in the `DB_RECOVERY_FILE_DEST` location when:

- The `ARC` or `LGWR` background process archives an online redo log or
- An `ALTER SYSTEM ARHIVE LOG CURRENT` statement is issued.

For example, assume that the following parameter settings are included in the initialization parameter file:

```
DB_RECOVERY_FILE_DEST_SIZE = 20G
DB_RECOVERY_FILE_DEST      = '/u01/oradata'
LOG_ARCHIVE_DEST_1         = 'LOCATION=USE_DB_RECOVERY_FILE_DEST'
```

Behavior of Oracle-Managed Files

The filenames of Oracle-managed files are accepted in SQL statements wherever a filename is used to identify an existing file. These filenames, like other filenames, are stored in the control file and, if using Recovery Manager (RMAN) for backup and recovery, in the RMAN catalog. They are visible in all of the usual fixed and dynamic performance views that are available for monitoring datafiles and tempfiles (for example, V$DATAFILE or DBA_DATA_FILES).

The following are some examples of statements using database-generated filenames:

```
SQL> ALTER DATABASE
  2> RENAME FILE '/u01/oradata/mydb/datafile/o1_mf_tbs01_ziw3bopb_.dbf'
  3> TO '/u01/oradata/mydb/tbs0101.dbf';

SQL> ALTER DATABASE
  2> DROP LOGFILE '/u01/oradata/mydb/onlinelog/o1_mf_1_wo94n2xi_.log';

SQL> ALTER TABLE emp
  2> ALLOCATE EXTENT
  3> (DATAFILE '/u01/oradata/mydb/datafile/o1_mf_tbs1_2ixfh90q_.dbf');
```

You can backup and restore Oracle-managed datafiles, tempfiles, and control files as you would corresponding non Oracle-managed files. Using database-generated filenames does not impact the use of logical backup files such as export files. This is particularly important for tablespace point-in-time recovery (TSPITR) and transportable tablespace export files.

There are some cases where Oracle-managed files behave differently. These are discussed in the sections that follow.

Dropping Datafiles and Tempfiles

Unlike files that are not managed by the database, when an Oracle-managed datafile or tempfile is dropped, the filename is removed from the control file and the file is automatically deleted from the file system. The statements that delete Oracle-managed files when they are dropped are:

- DROP TABLESPACE
- ALTER DATABASE TEMPFILE ... DROP

Dropping Redo Log Files

When an Oracle-managed redo log file is dropped its Oracle-managed files are deleted. You specify the group or members to be dropped. The following statements drop and delete redo log files:

- `ALTER DATABASE DROP LOGFILE`
- `ALTER DATABASE DROP LOGFILE MEMBER`

Renaming Files

The following statements are used to rename files:

- `ALTER DATABASE RENAME FILE`
- `ALTER TABLESPACE ... RENAME DATAFILE`

These statements do not actually rename the files on the operating system, but rather, the names in the control file are changed. If the old file is an Oracle-managed file and it exists, then it is deleted. You must specify each filename using the conventions for filenames on your operating system when you issue this statement.

Managing Standby Databases

The datafiles, control files, and redo log files in a standby database can be managed by the database. This is independent of whether Oracle-managed files are used on the primary database.

When recovery of a standby database encounters redo for the creation of a datafile, if the datafile is an Oracle-managed file, then the recovery process creates an empty file in the local default file system location. This allows the redo for the new file to be applied immediately without any human intervention.

When recovery of a standby database encounters redo for the deletion of a tablespace, it deletes any Oracle-managed datafiles in the local file system. Note that this is independent of the `INCLUDING DATAFILES` option issued at the primary database.

Scenarios for Using Oracle-Managed Files

This section further demonstrates the use of Oracle-managed files by presenting scenarios of their use.

Scenario 1: Create and Manage a Database with Multiplexed Redo Logs

In this scenario, a DBA creates a database where the datafiles and redo log files are created in separate directories. The redo log files and control files are multiplexed. The database uses an undo tablespace, and has a default temporary tablespace. The following are tasks involved with creating and maintaining this database.

1. Setting the initialization parameters

   The DBA includes three generic file creation defaults in the initialization parameter file before creating the database. Automatic undo management mode is also specified.

   ```
   DB_CREATE_FILE_DEST = '/u01/oradata'
   DB_CREATE_ONLINE_LOG_DEST_1 = '/u02/oradata'
   DB_CREATE_ONLINE_LOG_DEST_2 = '/u03/oradata'
   UNDO_MANAGEMENT = AUTO
   ```

   The DB_CREATE_FILE_DEST parameter sets the default file system directory for the datafiles and tempfiles.

   The DB_CREATE_ONLINE_LOG_DEST_1 and DB_CREATE_ONLINE_LOG_DEST_2 parameters set the default file system directories for redo log file and control file creation. Each redo log file and control file is multiplexed across the two directories.

2. Creating a database

   Once the initialization parameters are set, the database can be created by using this statement:

   ```
   SQL> CREATE DATABASE sample
   2>   DEFAULT TEMPORARY TABLESPACE dflttmp;
   ```

   Because a DATAFILE clause is not present and the DB_CREATE_FILE_DEST initialization parameter is set, the SYSTEM tablespace datafile is created in the default file system (/u01/oradata in this scenario). The filename is uniquely generated by the database. The file is autoextensible with an initial size of 100 MB and an unlimited maximum size. The file is an Oracle-managed file. A similar datafile is created for the SYSAUX tablespace.

   Because a LOGFILE clause is not present, two redo log groups are created. Each log group has two members, with one member in the DB_CREATE_ONLINE_LOG_DEST_1 location and the other member in the DB_CREATE_ONLINE_LOG_DEST_2 location. The filenames are uniquely generated by the database.

The log files are created with a size of 100 MB. The log file members are Oracle-managed files.

Similarly, because the CONTROL_FILES initialization parameter is not present, and two DB_CREATE_ONLINE_LOG_DEST_*n* initialization parameters are specified, two control files are created. The control file located in the DB_CREATE_ONLINE_LOG_DEST_1 location is the primary control file; the control file located in the DB_CREATE_ONLINE_LOG_DEST_2 location is a multiplexed copy. The filenames are uniquely generated by the database. They are Oracle-managed files. Assuming there is a server parameter file, a CONTROL_FILES initialization parameter is generated.

Automatic undo management mode is specified, but because an undo tablespace is not specified and the DB_CREATE_FILE_DEST initialization parameter is set, a default undo tablespace named SYS_UNDOTBS is created in the directory specified by DB_CREATE_FILE_DEST. The datafile is a 10 MB datafile that is autoextensible. It is an Oracle-managed file.

Lastly, a default temporary tablespace named dflttmp is specified. Because DB_CREATE_FILE_DEST is included in the parameter file, the tempfile for dflttmp is created in the directory specified by that parameter. The tempfile is 100 MB and is autoextensible with an unlimited maximum size. It is an Oracle-managed file.

The resultant file tree, with generated filenames, is as follows:

```
/u01
     /oradata
          /SAMPLE
               /datafile
                    /o1_mf_system_cmr7t30p_.dbf
                    /o1_mf_sysaux_cmr7t88p_.dbf
                    /o1_mf_sys_undotbs_2ixfh90q_.dbf
                    /o1_mf_dflttmp_157se6ff_.tmp
/u02
     /oradata
          /SAMPLE
               /onlinelog
                    /o1_mf_1_0orrm31z_.log
                    /o1_mf_2_2xyz16am_.log
               /controlfile
                    /o1_mf_cmr7t30p_.ctl
/u03
     /oradata
          /SAMPLE
```

```
                /onlinelog
                     /o1_mf_1_ixfvm8w9_.log
                     /o1_mf_2_q89tmp28_.log
                /controlfile
                     /o1_mf_x1sr8t36_.ctl
```

The internally generated filenames can be seen when selecting from the usual views. For example:

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

NAME
--------------------------------------------------
/u01/oradata/SAMPLE/datafile/o1_mf_system_cmr7t30p_.dbf
/u01/oradata/SAMPLE/datafile/o1_mf_sysaux_cmr7t88p_.dbf
/u01/oradata/SAMPLE/datafile/o1_mf_sys_undotbs_2ixfh90q_.dbf

3 rows selected
```

The name is also printed to the alert file when the file is created.

3. Managing control files

   The control file was created when generating the database, and a CONTROL_FILES initialization parameter was added to the parameter file. If needed, then the DBA can re-create the control file or build a new one for the database using the CREATE CONTROLFILE statement.

   The correct Oracle-managed filenames must be used in the DATAFILE and LOGFILE clauses. The ALTER DATABASE BACKUP CONTROLFILE TO TRACE statement generates a script with the correct filenames. Alternatively, the filenames can be found by selecting from the V$DATAFILE, V$TEMPFILE, and V$LOGFILE views. The following example re-creates the control file for the sample database:

```
CREATE CONTROLFILE REUSE
  DATABASE sample
  LOGFILE
    GROUP 1('/u02/oradata/SAMPLE/onlinelog/o1_mf_1_0orrm31z_.log',
            '/u03/oradata/SAMPLE/onlinelog/o1_mf_1_ixfvm8w9_.log'),
    GROUP 2('/u02/oradata/SAMPLE/onlinelog/o1_mf_2_2xyz16am_.log',
            '/u03/oradata/SAMPLE/onlinelog/o1_mf_2_q89tmp28_.log')
  NORESETLOGS
  DATAFILE '/u01/oradata/SAMPLE/datafile/o1_mf_system_cmr7t30p_.dbf',
           '/u01/oradata/SAMPLE/datafile/o1_mf_sysaux_cmr7t88p_.dbf',
           '/u01/oradata/SAMPLE/datafile/o1_mf_sys_undotbs_2ixfh90q_.dbf',
```

```
          '/u01/oradata/SAMPLE/datafile/o1_mf_dflttmp_157se6ff_.tmp'
  MAXLOGFILES 5
  MAXLOGHISTORY 100
  MAXDATAFILES 10
  MAXINSTANCES 2
  ARCHIVELOG;
```

The control file created by this statement is located as specified by the CONTROL_FILES initialization parameter that was generated when the database was created. The REUSE clause causes any existing files to be overwritten.

4. Managing the redo log

   To create a new group of redo log files, the DBA can use the ALTER DATABASE ADD LOGFILE statement. The following statement adds a log file with a member in the DB_CREATE_ONLINE_LOG_DEST_1 location and a member in the DB_CREATE_ONLINE_LOG_DEST_2 location. These files are Oracle-managed files.

   ```
   SQL> ALTER DATABASE ADD LOGFILE;
   ```

   Log file members continue to be added and dropped by specifying complete filenames.

   The GROUP clause can be used to drop a log group. In the following example the operating system file associated with each Oracle-managed log file member is automatically deleted.

   ```
   SQL> ALTER DATABASE DROP LOGFILE GROUP 3;
   ```

5. Managing tablespaces

   The default storage for all datafiles for future tablespace creations in the sample database is the location specified by the DB_CREATE_FILE_DEST initialization parameter (/u01/oradata in this scenario). Any datafiles for which no filename is specified, are created in the file system specified by the initialization parameter DB_CREATE_FILE_DEST. For example:

   ```
   SQL> CREATE TABLESPACE tbs_1;
   ```

   The preceding statement creates a tablespace whose storage is in /u01/oradata. A datafile is created with an initial of 100 MB and it is autoextensible with an unlimited maximum size. The datafile is an Oracle-managed file.

When the tablespace is dropped, the Oracle-managed files for the tablespace are automatically removed. The following statement drops the tablespace and all the Oracle-managed files used for its storage:

```
SQL> DROP TABLESPACE tbs_1;
```

Once the first datafile is full, the database does not automatically create a new datafile. More space can be added to the tablespace by adding another Oracle-managed datafile. The following statement adds another datafile in the location specified by DB_CREATE_FILE_DEST:

```
SQL> ALTER TABLESPACE tbs_1 ADD DATAFILE;
```

The default file system can be changed by changing the initialization parameter. This does not change any existing datafiles. It only affects future creations. This can be done dynamically using the following statement:

```
SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST='/u04/oradata';
```

6. Archiving redo information

   Archiving of redo log files is no different for Oracle-managed files, than it is for unmanaged files. A file system location for the archived log files can be specified using the LOG_ARCHIVE_DEST_*n* initialization parameters. The filenames are formed based on the LOG_ARCHIVE_FORMAT parameter or its default. The archived logs are not Oracle-managed files

7. Backup, restore, and recover

   Since an Oracle-managed file is compatible with standard operating system files, you can use operating system utilities to backup or restore Oracle-managed files. All existing methods for backing up, restoring, and recovering the database work for Oracle-managed files.

Scenario 2: Create and Manage a Database with Database Area and Flash Recovery Area

In this scenario, a DBA creates a database where the control files and redo log files are multiplexed. Archived logs and RMAN backups are created in the flash recovery area. The following tasks are involved in creating and maintaining this database:

1. Setting the initialization parameters

   The DBA includes the following generic file creation defaults:

```
DB_CREATE_FILE_DEST = '/u01/oradata'
DB_RECOVERY_FILE_DEST_SIZE = 10G
DB_RECOVERY_FILE_DEST = '/u02/oradata'
LOG_ARCHIVE_DEST_1 = 'LOCATION = USE_DB_RECOVERY_FILE_DEST'
```

The DB_CREATE_FILE_DEST parameter sets the default file system directory for datafiles, tempfiles, control files, and redo logs.

The DB_RECOVERY_FILE_DEST parameter sets the default file system directory for control files, redo logs, and RMAN backups.

The LOG_ARCHIVE_DEST_1 configuration 'LOCATION=USE_DB_RECOVERY_FILE_DEST' redirects archived logs to the DB_RECOVERY_FILE_DEST location.

The DB_CREATE_FILE_DEST and DB_RECOVERY_FILE_DEST parameters set the default directory for log file and control file creation. Each redo log and control file is multiplexed across the two directories.

2. Creating a database
3. Managing control files
4. Managing the redo log
5. Managing tablespaces

   Tasks 2, 3, 4, and 5 are the same as in Scenario 1, except that the control files and redo logs are multiplexed across the DB_CREATE_FILE_DEST and DB_RECOVERY_FILE_DEST locations.

6. Archiving redo log information

   Archiving online logs is no different for Oracle-managed files than it is for unmanaged files. The archived logs are created in DB_RECOVERY_FILE_DEST and are Oracle-managed files.

7. Backup, restore, and recover

   An Oracle-managed file is compatible with standard operating system files, so you can use operating system utilities to backup or restore Oracle-managed files. All existing methods for backing up, restoring, and recovering the database work for Oracle-managed files. When no format option is specified, all disk backups by RMAN are created in the DB_RECOVERY_FILE_DEST location. The backups are Oracle-managed files.

Scenario 3: Adding Oracle-Managed Files to an Existing Database

Assume in this case that an existing database does not have any Oracle-managed files, but the DBA would like to create new tablespaces with Oracle-managed files and locate them in directory `/u03/oradata`.

1. Setting the initialization parameters

   To allow automatic datafile creation, set the `DB_CREATE_FILE_DEST` initialization parameter to the file system directory in which to create the datafiles. This can be done dynamically as follows:

   ```
   SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST = '/u03/oradata';
   ```

2. Creating tablespaces

   Once `DB_CREATE_FILE_DEST` is set, the `DATAFILE` clause can be omitted from a `CREATE TABLESPACE` statement. The datafile is created in the location specified by `DB_CREATE_FILE_DEST` by default. For example:

   ```
   SQL> CREATE TABLESPACE tbs_2;
   ```

   When the `tbs_2` tablespace is dropped, its datafiles are automatically deleted.

12

Using Automatic Storage Management

This chapter briefly discusses some of the concepts behind Automatic Storage Management and describes how to use it. It contains the following topics:

What Is Automatic Storage Management?

Automatic Storage Management (ASM) simplifies database administration. It eliminates the need for you, as a DBA, to directly manage potentially thousands of Oracle database files. It does this by enabling you to create **disk groups**, which are comprised of disks and the files that reside on them. You only need to manage a small number of disk groups.

In the SQL statements that you use for creating database structures such as tablespaces, redo log and archive log files, and control files, you specify file location in terms of disk groups. Automatic Storage Management then creates and manages the associated underlying files for you.

Automatic Storage Management extends the power of Oracle-managed files. With Oracle-managed files, files are created and managed automatically for you, but with Automatic Storage Management you get the additional benefits of features such as mirroring and striping.

Automatic Storage Management does not eliminate any existing database functionality. Existing databases are able to operate as they always have. Existing

databases using file systems or with storage on raw devices can operate as they always have. New files can be created as ASM files while old ones are administered in the old way. Databases can have a mixture of ASM files, Oracle-managed files, and manually managed files all at the same time.

To create a database that uses storage managed by Automatic Storage Management, you must first start an ASM instance. However, an ASM instance does not require that a database instance is running

Automatic Storage Management is integrated into the database server; you do not need to install it as a separate product.

> **See Also:** *Oracle Database Concepts* for an overview of Automatic Storage Management

Overview of the Components of Automatic Storage Management

The primary component of Automatic Storage Management is the **disk group**. You configure Automatic Storage Management by creating disk groups, which, in your database instance, can then be specified as the default location for files created in the database. Oracle provides SQL statements that create and manage disk groups, their contents, and their metadata.

A disk group consists of a grouping of disks that are managed together as a unit. These disks are referred to as **ASM disks**. Files written on ASM disks are **ASM files**, whose names are automatically generated by Automatic Storage Management. You can specify user-friendly **alias names** for ASM files, but you must create a hierarchical **directory** structure for these alias names.

You can affect how Automatic Storage Management places files on disks by specifying **failure groups**. Failure groups define disks that share components, such that if one fails then other disks sharing the component might also fail. An example of what you might define as a failure group would be a set of SCSI disks sharing the same SCSI controller. Failure groups are used to determine which ASM disks to use for storing redundant data. For example, if two-way mirroring is specified for a file, then redundant copies of file extents must be stored in separate failure groups.

Templates are used to provide the attribute information about ASM files created in ASM disk groups. These templates simplify file creation by mapping complex file attribute specifications into a single name. For example, a template named `ONLINELOG` provides the file redundancy and striping attributes for all redo log files written to ASM disks. For each disk group, Automatic Storage Management provides a set of initial system templates, as shown in the table in "Managing Disk

Group Templates" on page 12-21, suitable for most needs, but you can modify the templates or create new ones to meet unique requirements.

Administering an ASM Instance

> **Note:** You can use Oracle Enterprise Manager (EM) or the Database Configuration Assistant (DBCA) for a GUI interface to Automatic Storage Management that replaces the use of SQL or SQL*Plus for configuring and altering disk groups and their metadata.
>
> DBCA eases the configuring and creation of your database while EM provides an integrated approach for managing both your ASM instance and database instance.

The functionality of an ASM instance can be summarized as follows:

- It manages groups of disks, called disk groups.
- It protects the data within a disk group.
- It provides near-optimal I/O balancing without any manual tuning.
- It enables the user to manage database objects such as tablespaces without needing to specify and track filenames.
- It supports large files.

This section discusses the administration of an ASM instance in terms of its startup and shutdown, and its behavior and interaction with the database instance. The major part of the administration of an ASM instance, once it is started, is the creation and maintenance of disk groups and their metadata. This is discussed in "Configuring the Components of Automatic Storage Management" on page 12-10.

This section contains the following topics:

- Installation of ASM
- Authentication for Accessing an ASM Instance
- Setting Initialization Parameters for an ASM Instance
- Starting Up an ASM Instance
- Shutting Down an ASM Instance

- Disk Group Fixed Tables

Installation of ASM

Automatic Storage Management is always installed by the Oracle Universal Installer when you install your database software. The Database Configuration Assistant (DBCA) determines if an ASM instance already exists, and if not, then you are given the option of creating and configuring an ASM instance as part of database creation and configuration. If an ASM instance already exists, then it is used instead.

DBCA also configures your instance parameter file and password file.

Authentication for Accessing an ASM Instance

Automatic Storage Management security considerations derive from the fact that a particular ASM instance is tightly bound to one or more database instances operating on the same server. In effect, the ASM instance is a logical extension of these database instances. Both the ASM instance and the database instances must have equivalent operating system access rights (read / write) to the disk group member disks. For UNIX this is typically provided through shared UNIX group membership.

ASM instances do not have a data dictionary, so the only way to connect to one is as an administrator. This means you use operating system authentication and connect as SYSDBA or SYSOPER, or to connect remotely, use a password file.

Using operating system authentication, the authorization to connect with the SYSDBA privilege is granted through the use of an operating system group. On UNIX, this is typically the dba group. By default, members of the dba group are authorized to connect with the SYSDBA privilege on all instances on the node, including the ASM instance. Users who connect to the ASM instance with SYSDBA privilege have complete administrative access to all disk groups that are managed by that ASM instance.

> **See Also:** "Database Administrator Authentication" on page 1-12 for information about OS and password file authentication as relating to connecting to a database instance. Administrators are similarly authenticated and connected for an ASM instance.

Setting Initialization Parameters for an ASM Instance

Some initialization parameters are specifically relevant to an ASM instance. Of those initialization parameters intended for a database instance, only a few are relevant to an ASM instance. You can set those parameters at database creation time using Database Configuration Assistant or later using Enterprise Manager. The remainder of this section describes setting the parameters manually by editing the initialization parameter file.

Initialization Parameters for ASM Instances

The following initialization parameters relate to an ASM instance. Parameters that start with ASM_ cannot be set in database instances.

Name	Description
INSTANCE_TYPE	Must be set to INSTANCE_TYPE = ASM. **Note:** This is the only required parameter. All other parameters take suitable defaults for most environments.
DB_UNIQUE_NAME	Unique name for this group of ASM instances within the cluster or on a node. Default: +ASM (Needs to be modified only if trying to run multiple ASM instances on the same node)
ASM_POWER_LIMIT	The maximum power on an ASM instance for disk rebalancing. Default: 1 **See Also:** "Tuning Rebalance Operations" on page 12-6
ASM_DISKSTRING	Limits the set of disks that Automatic Storage Management considers for discovery. Default: NULL (This default causes ASM to find all of the disks in a platform-specific location to which it has read / write access.) **See Also:** "Improving Disk Discovery Time" on page 12-7

Name	Description
ASM_DISKGROUPS	Lists the names of disk groups to be mounted by an ASM instance at startup, or when the ALTER DISKGROUP ALL MOUNT statement is used. Default: NULL (If this parameter is not specified, then no disk groups are mounted.) **Note:** This parameter is dynamic and if you are using a server parameter file (SPFILE), then you should rarely need to manually alter this value. Automatic Storage Management automatically adds a disk group to this parameter when a disk group is successfully mounted, and automatically removes a disk group that is specifically dismounted. However, when using a traditional text initialization parameter file, remember to edit the initialization parameter file to add the name of any disk group that you want automatically mounted at instance startup, and remove the name of any disk group that you no longer want automatically mounted.

Tuning Rebalance Operations

Oracle Database can perform one rebalance at a time on a given instance. The rebalance power is constrained by the value of the ASM_POWER_LIMIT initialization parameter. You can adjust this parameter dynamically. The higher the limit, the faster a rebalance operation may complete. Lower values cause rebalancing to take longer, but consume fewer processing and I/O resources. If the POWER clause is not specified, or when rebalance is implicitly invoked by adding or dropping a disk, the power defaults to the ASM_POWER_LIMIT.

The V$ASM_OPERATION view provides information that can be used for adjusting ASM_POWER_LIMIT and the resulting power of rebalance operations. If the DESIRED_POWER column value is less than the ACTUAL_POWER column value for a given rebalance operation, then ASM_POWER_LIMIT is impacting the rebalance.

The V$ASM_OPERATION view also gives an estimate in the EST_MINUTES column of the amount of time remaining for the operation to complete. You can see the effect of changing the power of rebalance by observing the change in the time estimate.

If a rebalance is in progress because a failed disk is being dropped, increasing the power of the rebalance shortens the window during which redundant copies on the failed disk are being reconstructed on other disks. Lowering ASM_POWER_LIMIT reduces the amount of CPU and I/O bandwidth that Automatic Storage Management consumes for a rebalance. This leaves these resources available for other applications, such as the database.

The default value errors on the side of minimizing disruption to other applications. The appropriate value is dependent upon your hardware configuration as well as performance and availability requirements.

Improving Disk Discovery Time

The value for the ASM_DISKSTRING initialization parameter is an operating system dependent value used by Automatic Storage Management to limit the set of disks considered for discovery. When a new disk is added to a disk group, each ASM instance that has the disk group mounted must be able to discover the new disk using its ASM_DISKSTRING.

In many cases, the default value is sufficient. Using a more restrictive value may reduce the time required for Automatic Storage Management to perform discovery, and thus improve disk group mount time or the time for adding a disk to a disk group. It may be necessary to dynamically change the ASM_DISKSTRING before adding a disk so that the new disk will be discovered through this parameter.

If your site is using a third-party vendor ASMLIB, that vendor may have discovery string conventions that should be used for ASM_DISKSTRING.

Behavior of Database Initialization Parameters in an ASM Instance

If you specify a database instance initialization parameter in an ASM initialization parameter file, it can have one of three effects:

- If the parameter is not valid in the ASM instance, it produces a ORA-15021 error.
- Some database parameters can be specified and are valid in an ASM instance, for example those relating to dump destinations and some buffer cache parameters. In general, Oracle will select appropriate defaults for any database parameters that are relevant to the ASM instance.

If you specify any of the ASM specific parameters (names start with ASM_) in a database instance parameter file, you will receive an ORA-15021 error.

Starting Up an ASM Instance

ASM instances are started similarly to Oracle database instances with some minor differences. These are:

- The initialization parameter file, which can be a server parameter file, must contain:

```
INSTANCE_TYPE = ASM
```

This parameter signals the Oracle executable that an ASM instance is starting and not a database instance.

Using a server parameter file is recommended because it eliminates the need to make manual changes to a text initialization parameter file.

- For ASM instances, STARTUP tries to mount the disk groups specified by the initialization parameter ASM_DISKGROUPS and not the database.

Further, the SQL*Plus STARTUP command parameters are interpreted by Automatic Storage Management as follows:

STARTUP Parameter	Description
FORCE	Issues a SHUTDOWN ABORT to the ASM instance before restarting it
MOUNT	Mounts the disk groups specified in the ASM_DISKGROUPS initialization parameter
NOMOUNT	Starts up the ASM instance without mounting any disk groups
OPEN	Invalid for an ASM instance

The following is a sample SQL*Plus session where an ASM instance is started:

```
% sqlplus /nolog
SQL> CONNECT / AS sysdba
Connected to an idle instance.
SQL> STARTUP
ASM instance started
Total System Global Area 147936196 bytes
Fixed Size 324548 bytes
Variable Size 96468992 bytes
Database Buffers 50331648 bytes
Redo Buffers 811008 bytes
ASM diskgroups mounted
```

ASM Instance Memory Requirements

ASM instances are smaller than database instances. A 64 MB SGA should be sufficient for all but the largest ASM installations.

Disk Discovery

When an ASM instance initializes, ASM is able to discover and look at the contents of all of the disks in the disk groups that are pointed to by the `ASM_DISKSTRING` initialization parameter. This saves you from having to specify a path for each of the disks in the disk group.

Disk group mounting requires that an ASM instance doing disk discovery be able to access all the disks within the disk group that any other ASM instance having previously mounted the disk group believes are members of that disk group. It is vital that any disk configuration errors be detected before a disk group is mounted.

Automatic Storage Management attempts to identify the following configuration errors:

1. A single disk with different mount points is presented to an ASM instance. This can be caused by multiple paths to a single disk. In this case, if the disk in question is part of a disk group, disk group mount fails. If the disk is being added to a disk group with `ADD DISK` or `CREATE DISKGROUP`, the command fails. To correct the error, restrict the disk string so that it does not include multiple paths to the same disk.
2. Multiple ASM disks, with the same ASM label, passed to separate ASM instances as the same disk. In this case, disk group mount fails.
3. Disks that were not intended to be ASM disks are passed to an ASM instance by the discovery function. ASM does not overwrite a disk if it recognizes the header as that of an Oracle object.

Disk Group Recovery

When an ASM instance fails, then all Oracle database instances on the same node as that ASM instance and that use a disk group managed by that ASM instance also fail. In a single ASM instance configuration, if the ASM instance fails while ASM metadata is open for update, then after the ASM instance reinitializes, it reads the disk group log and recovers all transient changes.

With multiple ASM instances sharing disk groups, if one ASM instance should fail, another ASM instance automatically recovers transient ASM metadata changes caused by the failed instance. The failure of an Oracle database instance is not significant here because only ASM instances update ASM metadata.

Shutting Down an ASM Instance

Using SQL*Plus, Automatic Storage Management shutdown is initiated by issuing the SHUTDOWN command. For example:

```
% sqlplus /nolog
SQL> CONNECT / AS sysdba
Connected.
SQL> SHUTDOWN NORMAL
```

The table that follows lists the SHUTDOWN modes and describes the behavior of the ASM instance in each mode.

SHUTDOWN Mode	Action Taken By Automatic Storage Management
NORMAL	ASM waits for the connected ASM instances (and other ASM SQL sessions) to exit before shutting down the instance.
IMMEDIATE	ASM waits for any in-progress SQL to complete before shutting down the ASM instance, but does not wait for database instances to disconnect.
TRANSACTIONAL	Same as IMMEDIATE.
ABORT	Automatic Storage Management immediately shuts down.

Disk Group Fixed Tables

A number of fixed tables, visible from both the ASM and database instances are provided for administrative and debugging purposes. These views are discussed in "Viewing Information About Automatic Storage Management" on page 12-44.

Configuring the Components of Automatic Storage Management

This section starts by presenting a brief overview of the components of Automatic Storage Management and discusses some considerations and guidelines that you should be aware of as you configure your ASM instance. Then, specific operations that you use to configure and maintain the ASM instance are discussed.

If you have a database instance running and are actively using Automatic Storage Management, you can keep the database open and running while you reconfigure disk groups.

The SQL statements introduced in this section are only available in an ASM instance. You must first start the ASM instance. This is discussed in "Administering an ASM Instance" on page 12-3.

The following topics are contained in this section:

Considerations and Guidelines for Configuring an ASM Instance

The following are some considerations and guidelines to be aware of as you configure Automatic Storage Management.

Determining the Number of Disk Groups

The following criteria can help you determine the number of disk groups that you create:

- Disks in a given disk group should have similar size and performance characteristics. If you have several different types of disks in terms of size and performance, then it would be better to form several disk groups accordingly.
- For recovery reasons, you might feel more comfortable having separate disk groups for your database files and flash recovery area files. Using this approach, even with the loss of one disk group, the database would still be intact.

Storage Arrays and Automatic Storage Management

With Automatic Storage Management, the definition of the logical volumes of a storage array is critical to database performance. Automatic Storage Management cannot optimize database data placement when the storage array disks are subdivided or aggregated. Aggregating and subdividing the physical volumes of an array into logical volumes can hide the physical disk boundaries from Automatic

Storage Management. Consequently, careful consideration of storage array configuration is required.

Consider Performance Characteristics when Grouping Disks

Automatic Storage Management eliminates the need for manual disk tuning. However, to ensure consistent performance, you should avoid placing dissimilar disks in the same disk group. For example, the newest and fastest disks might reside in a disk group reserved for the database work area, and slower drives could reside in a disk group reserved for the flash recovery area.

Automatic Storage Management load balances file activity by uniformly distributing file extents across all disks in a disk group. For this technique to be effective it is important that the disks used in a disk group be of similar performance characteristics.

There may be situations where it is acceptable to temporarily have disks of different sizes and performance co-existing in a disk group. This would be the case when migrating from an old set of disks to a new set of disks. The new disks would be added and the old disks dropped. As the old disks are dropped, their storage is migrated to the new disks while the disk group in online.

Effects of Adding and Dropping Disks from a Disk Group

Automatic Storage Management automatically rebalances--distributes file data evenly across all the disks of a disk group--whenever disks are added or dropped. ASM allocates files in such a way that rebalancing is not required when the number of disks is static. A disk is not released from a disk group until data is moved off of the disk through rebalancing. Likewise, a newly added disk cannot support its share of the I/O workload until rebalancing completes. It is more efficient to add or drop multiple disks at the same time so that they are rebalanced as a single operation. This avoids unnecessary movement of data.

You can add or drop disks without shutting down the database. However, a performance impact on I/O activity may result.

Failure Groups and Mirroring

Mirroring of metadata and user data is achieved through failure groups. ASM requires at least two failure groups for normal redundancy disk groups and at least three failure groups for high redundancy disk groups. System reliability can be hampered if an insufficient number of failure groups are provided. Consequently, failure group configuration is very important to creating a highly reliable system.

Scalability

ASM imposes the following limits:

- 63 disk groups in a storage system
- 10,000 ASM disks in a storage system
- 4 petabyte maximum storage for each ASM disk
- 40 exabyte maximum storage for each storage system
- 1 million files for each disk group
- 2.4 terabyte maximum storage for each file

Creating a Disk Group

You use the `CREATE DISKGROUP` statement to create disk groups. This statement enables you to assign a name to the disk group and to specify the disks that are to be formatted as ASM disks belonging to the disk group. You specify the disks as one or more operating system dependent search strings that Automatic Storage Management then uses to find the disks.

You can specify the disks as belonging to specific failure groups, and you can specify the redundancy level for the disk group. If you do not specify a disk as belonging to a failure group, that disk comprises its own failure group. The redundancy level can be specified as `NORMAL REDUNDANCY` or `HIGH REDUNDANCY`, as defined by the disk group templates. You can also specify `EXTERNAL REDUNDANCY` for external redundancy disk groups, which do not have failure groups. You might do this if you want to use storage array protection features instead.

Automatic Storage Management programmatically determines the size of each disk. If for some reason this is not possible, or if you want to restrict the amount of space used on a disk, you are able to specify a `SIZE` clause for each disk. Automatic Storage Management creates operating system independent names for the disks in a disk group that you can use to reference the disks in other SQL statements. Optionally, you can provide your own name for a disk using the `NAME` clause.

The ASM instance ensures that any disk being included in a disk group is addressable and usable. This requires reading the first block of the disk to determine if it already belongs in a disk group. If not, a header is written. It is not possible for a disk to be a member of multiple disk groups.

However, you can force a disk that is already a member of another disk group to become a member of the disk group you are creating by specifying the `FORCE`

clause. For example, a disk with an ASM header might have failed temporarily, so that its header could not be cleared when it was dropped from its disk group. Once the disk is repaired, it is no longer part of any disk group, but it still has an ASM header. The FORCE flag is required to use the disk in a new disk group. The original disk group must not be mounted, and the disk must have a disk group header, otherwise the operation fails. Note that if you do this, you may cause another disk group to become unusable. If you specify NOFORCE, which is the default, you receive an error if you attempt to include a disk that already belongs to another disk group.

The CREATE DISKGROUP statement mounts the disk group for the first time, and adds the disk group name to the ASM_DISKGROUPS initialization parameter if a server parameter file is being used. If a text initialization parameter file is being used and you want the disk group to be automatically mounted at instance startup, then you must remember to add the disk group name to the ASM_DISKGROUPS initialization parameter before the next time that you shut down and restart the ASM instance.

Creating a Disk Group: Example

The following examples assume that the ASM_DISKSTRING is set to '/devices/*'. Assume the following:

- ASM disk discovery identifies the following disks in directory /devices.

  ```
  /devices/diska1
  /devices/diska2
  /devices/diska3
  /devices/diska4
  /devices/diskb1
  /devices/diskb2
  /devices/diskb3
  /devices/diskb4
  ```

- The disks diska1 - diska4 are on a separate SCSI controller from disks diskb1 - diskb4.

The following SQL*Plus session illustrates starting an ASM instance and creating a disk group named dgroup1.

```
% SQLPLUS /NOLOG
SQL> CONNECT / AS SYSDBA
Connected to an idle instance.
SQL> STARTUP NOMOUNT
SQL> CREATE DISKGROUP dgroup1 NORMAL REDUNDANCY
```

```
 2  FAILGROUP controller1 DISK
 3 '/devices/diska1',
 4 '/devices/diska2',
 5 '/devices/diska3',
 6 '/devices/diska4',
 7 FAILGROUP controller2 DISK
 8 '/devices/diskb1',
 9 '/devices/diskb2',
10 '/devices/diskb3',
11 '/devices/diskb4';
```

In this example, dgroup1 is composed of eight disks that are defined as belonging to either failure group controller1 or controller2. Since NORMAL REDUNDANCY level is specified for the disk group, then Automatic Storage Management provides redundancy for all files created in dgroup1 according to the attributes specified in the disk group templates.

For example, in the system default template shown in the table in "Managing Disk Group Templates" on page 12-21, normal redundancy for the online redo log files (ONLINELOG template) is two-way mirroring. This means that when one copy of a redo log file extent is written to a disk in failure group controller1, a mirrored copy of the file extent is written to a disk in failure group controller2. You can see that to support normal redundancy level, at least two failure groups must be defined.

Since no NAME clauses are provided for any of the disks being included in the disk group, the disks are assigned the names of dgroup1_0001, dgroup1_0002, ..., dgroup1_0008.

Altering the Disk Membership of a Disk Group

At a later time after the creation of a disk group, you can change its composition by adding more disks, resizing disks, or dropping disks. You use clauses of the ALTER DISKGROUP statement to perform these actions. You can perform multiple operations with one ALTER DISKGROUP statement.

Automatic Storage Management automatically rebalances file extents when the composition of a disk group changes. Because rebalancing can be a long running operation, the ALTER DISKGROUP statement does not wait until the operation is complete before returning. To monitor progress of these long running operations, query the V$ASM_OPERATION dynamic performance view.

Adding Disks to a Disk Group

The ADD clause of the ALTER DISKGROUP statement lets you add disks to a disk group, or to add a failure group to the disk group. The ALTER DISKGROUP clauses that you can use when adding disks to a disk group are similar to those that can be used when specifying the disks to be included when initially creating a disk group. This is discussed in "Creating a Disk Group" on page 12-13.

The new disks will gradually start to carry their share of the workload as rebalancing progresses.

Automatic Storage Management behavior when adding disks to a disk group is best illustrated through examples.

Adding Disks to a Disk Group: Example 1 The following statement adds disks to dgroup1:

```
ALTER DISKGROUP dgroup1 ADD DISK
     '/devices/diska5' NAME diska5,
     '/devices/diska6' NAME diska6,
     '/devices/diska7' NAME diska7,
     '/devices/diska8' NAME diska8;
```

Since no FAILGROUP clauses are included in the ALTER DISKGROUP statement, each disk is assigned to its own failgroup. The NAME clauses assign names to the disks, otherwise they would have been assigned system generated names.

Adding Disks to a Disk Group: Example 2 The statements presented in this example demonstrate the interactions of disk discovery with the ADD DISK operation.

Assume that disk discovery now identifies the following disks in directory /devices:

/devices/diska1 -- member of dgroup1
/devices/diska2 -- member of dgroup1
/devices/diska3 -- member of dgroup1
/devices/diska4 -- member of dgroup1
/devices/diska5 -- candidate disk
/devices/diska6 -- candidate disk
/devices/diska7 -- candidate disk
/devices/diska8 -- candidate disk
/devices/diskb1 -- member of dgroup1
/devices/diskb2 -- member of dgroup1
/devices/diskb3 -- member of dgroup1

/devices/diskb4 -- member of dgroup1
/devices/diskc1 -- member of dgroup2
/devices/diskc2 -- member of dgroup2
/devices/diskc3 -- member of dgroup3
/devices/diskc4 -- candidate disk
/devices/diskd1 -- candidate disk
/devices/diskd2 -- candidate disk
/devices/diskd3 -- candidate disk
/devices/diskd4 -- candidate disk

Issuing the following statement adds disks /devices/diska5 - /devices/diska8 to dgroup1. It ignores /devices/diska1 - /devices/diska4 because they already belong to dgroup1, but does not fail because the other disks that match the search string are not already members of any other disk group.

```
ALTER DISKGROUP dgroup1 ADD DISK
     '/devices/diska*';
```

The following statement will fail, since the /devices/diska2 search string only matches a disk that already belongs to dgroup1:

```
ALTER DISKGROUP dgroup1 ADD DISK
     '/devices/diska2',
     '/devices/diskd4';
```

The following statement to add disks to dgroup1 will fail because the search string matches a disk that is contained in another disk group. Specifically, /devices/diskc1 belongs to disk group dgroup2.

```
ALTER DISKGROUP dgroup1 ADD DISK
     '/devices/disk*1';
```

The following statement succeeds in adding /devices/diskc4 and /devices/diskd1 - /devices/diskd4 to disk group dgroup1. It does not matter that /devices/diskd4 is included in both search strings (or that /devices/diska4 and /devices/diskb4 are already members of dgroup1).

```
ALTER DISKGROUP dgroup1 ADD DISK
      '/devices/disk*4',
      '/devices/diskd*';
```

The following use of the FORCE clause allows /devices/diskc3 to be added to dgroup2, even though it is a current member of dgroup3.

```
ALTER DISKGROUP dgroup2 ADD DISK
     '/devices/diskc3' FORCE;
```

For this statement to succeed, dgroup3 cannot be mounted.

Dropping Disks from Disk Groups

To drop disks from a disk group, use the DROP DISK clause of the ALTER DISKGROUP statement. You can also drop all of the disks in specified failure groups using the DROP DISKS IN FAILGROUP clause.

When a disk is dropped, the disk group is rebalanced by moving all of the file extents from the dropped disk to other disks in the disk group. The header on the dropped disk is cleared. If you specify the FORCE clause for the drop operation, the disk is dropped even if Automatic Storage Management cannot read or write to the disk. You cannot use the FORCE flag when dropping a disk from an external redundancy disk group.

Dropping Disks from Disk Groups: Example 1 This example drops diska5 (the operating system independent name assigned to /devices/c0t4d0s2 in "Adding Disks to a Disk Group: Example 1" on page 12-16) from disk group dgroup1.

```
ALTER DISKGROUP dgroup1 DROP DISK diska5;
```

Dropping Disks from Disk Groups: Example 2 This example also shows dropping diska5 from disk group dgroup1, but it illustrates how multiple actions are possible with one ALTER DISKGROUP statement.

```
ALTER DISKGROUP dgroup1 DROP DISK diskA5
     ADD FAILGROUP failgrp1 DISK '/devices/diska9' NAME diska9;
```

Resizing Disks in Disk Groups

The RESIZE clause of ALTER DISKGROUP enables you to perform the following operations:

- Resize all disks in the disk group
- Resize specific disks
- Resize all of the disks in a specified failure group

If you do not specify a new size in the SIZE clause then Automatic Storage Management uses the size of the disk as returned by the operating system. This could be a means of recovering disk space when you had previously restricted the size of the disk by specifying a size smaller than disk capacity.

The new size is written to the ASM disk header record and if the size of the disk is increasing, then the new space is immediately available for allocation. If the size is decreasing, rebalancing must relocate file extents beyond the new size limit to available space below the limit. If the rebalance operation can successfully relocate all extents, then the new size is made permanent, otherwise the rebalance fails.

Resizing Disks in Disk Groups: Example The following example resizes all of the disks in failgroup `failgrp1` of disk group `dgroup1`. If the new size is greater than disk capacity, the statement will fail.

```
ALTER DISKGROUP dgroup1
     RESIZE DISKS IN FAILGROUP failgrp1 SIZE 100G;
```

Undropping Disks in Disk Groups

The `UNDROP DISKS` clause of the `ALTER DISKGROUP` statement enables you to cancel all pending drops of disks within disk groups. If a drop disk operation has already completed, then this statement cannot be used to restore it. This statement cannot be used to restore disks that are being dropped as the result of a `DROP DISKGROUP` statement, or for disks that are being dropped using the `FORCE` clause.

Undropping Disks in Disk Groups: Example The following example cancels the dropping of disks from disk group dgroup1:

```
ALTER DISKGROUP dgroup1 UNDROP DISKS;
```

Manually Rebalancing a Disk Group

You can manually rebalance the files in a disk group using the `REBALANCE` clause of the `ALTER DISKGROUP` statement. This would normally not be required, since Automatic Storage Management automatically rebalances disk groups when their composition changes. You might want to do a manual rebalance operation if you want to control the speed of what would otherwise be an automatic rebalance operation.

The `POWER` clause of the `ALTER DISKGROUP ... REBALANCE` statement specifies the degree of parallelization, and thus the speed of the rebalance operation. It can be set to a value from 0 to 11, where a value of 0 stops rebalancing and a value of 11 (the default) causes rebalancing to occur as fast as possible. The power level of an ongoing rebalance operation can be changed by entering the rebalance statement with a new level. If you specify a power level of 0, rebalancing is halted until the statement is either implicitly or explicitly reinvoked.

The dynamic initialization parameter ASM_POWER_LIMIT specifies a limit on the degree of parallelization for rebalance operations. Even if you specify a higher value in the POWER clause, the degree of parallelization will not exceed the value specified by the ASM_POWER_LIMIT parameter.

The ALTER DISK GROUP ... REBALANCE statement uses the resources of the single node upon which it is started. ASM can perform one rebalance at a time on a given instance. The rebalance power is constrained by the value of the ASM_POWER_LIMIT initialization parameter.

You can query the V$ASM_OPERATION view for help adjusting ASM_POWER_LIMIT and the resulting power of rebalance operations. If the DESIRED_POWER column value is less than the ACTUAL_POWER column value for a given rebalance operation, then ASM_POWER_LIMIT is impacting the rebalance. The EST_MINUTES column indicates the amount of time remaining for the operation to complete. You can see the effect of changing the power of rebalance by observing the change in the time estimate.

Rebalancing continues across a failure of the ASM instance performing the rebalance.

See Also: "Tuning Rebalance Operations" on page 12-6

Manually Rebalancing a Disk Group: Example The following example manually rebalances the disk group dgroup2:

```
ALTER DISKGROUP dgroup2 REBALANCE POWER 5;
```

Mounting and Dismounting Disk Groups

Disk groups that are specified in the ASM_DISKGROUPS initialization parameter are mounted automatically at ASM instance startup. This makes them available to all database instances running on the same node as Automatic Storage Management. The disk groups are dismounted at ASM instance shutdown. Automatic Storage Management also automatically mounts a disk group when you initially create it, and dismounts a disk group if you drop it.

There may be times that you want to mount or dismount disk groups manually. For these actions use the ALTER DISKGROUP ... MOUNT or ALTER DISKGROUP ... DISMOUNT statement. You can mount or dismount disk groups by name, or specify ALL.

If you try to dismount a disk group that contains open files, the statement will fail, unless you also specify the FORCE clause.

Dismounting Disk Groups: Example

The following statement dismounts all disk groups that are currently mounted to the ASM instance:

```
ALTER DISKGROUP ALL DISMOUNT;
```

Mounting Disk Groups: Example

The following statement mounts disk group `dgroup1`:

```
ALTER DISKGROUP dgroup1 MOUNT;
```

Managing Disk Group Templates

A template is a named collection of attributes that are applied to files created within a disk group. Oracle provides a set of initial system default templates that Automatic Storage Management associates with a disk group when it is created. The table that follows lists the system default templates and the attributes they apply to the various file types that Automatic Storage Management supports.

You can add new templates to a disk group, change existing ones, or drop templates using clauses of the `ALTER DISKGROUP` statement. The `V$ASM_TEMPLATE` view lists all of the templates known to the ASM instance.

Table 12–1 Automatic Storage Management System Default File Group Templates

Template Name	File Type	External Redundancy	Normal Redundancy	High Redundancy	Striped
CONTROL	Control files	Unprotected	2-way mirror	3-way mirror	Fine
DATAFILE	Datafiles and copies	Unprotected	2-way mirror	3-way mirror	Coarse
ONLINELOG	Online logs	Unprotected	2-way mirror	3-way mirror	Fine
ARCHIVELOG	Archive logs	Unprotected	2-way mirror	3-way mirror	Coarse
TEMPFILE	Tempfiles	Unprotected	2-way mirror	3-way mirror	Coarse
BACKUPSET	Datafile backup pieces, datafile incremental backup pieces, and archive log backup pieces	Unprotected	2-way mirror	3-way mirror	Coarse
PARAMETERFILE	SPFILEs	Unprotected	2-way mirror	3-way mirror	Coarse
DATAGUARDCONFIG	Disaster recovery configurations (used in standby databases)	Unprotected	2-way mirror	3-way mirror	Coarse

Table 12–1 (Cont.) Automatic Storage Management System Default File Group Templates

Template Name	File Type	External Redundancy	Normal Redundancy	High Redundancy	Striped
FLASHBACK	Flashback logs	Unprotected	2-way mirror	3-way mirror	Fine
CHANGETRACKING	Block change tracking data (used during incremental backups)	Unprotected	2-way mirror	3-way mirror	Coarse
DUMPSET	Data Pump dumpset	Unprotected	2-way mirror	3-way mirror	Coarse
XTRANSPORT	Cross-platform converted datafile	Unprotected	2-way mirror	3-way mirror	Coarse
AUTOBACKUP	Automatic backup files	Unprotected	2-way mirror	3-way mirror	Coarse

Adding Templates to a Disk Group

To add a new template for a disk group use the ADD TEMPLATE clause of the ALTER DISKGROUP statement. You specify the name of the template, its redundancy attributes, and its striping attribute.

Adding Templates to a Disk Group: Example 1 The following statement creates a new template named reliable:

```
ALTER DISKGROUP dgroup2 ADD TEMPLATE reliable ATTRIBUTES (MIRROR FINE);
```

This statement creates a template that applies the following attributes to files:

Template Name	External Redundancy	Normal Redundancy	High Redundancy	Striping
RELIABLE	You cannot specify RELIABLE for an external redundancy group	2-way mirror	3-way mirror	Every 128 KB

Adding Templates to a Disk Group: Example 2 This statement creates a new template named unreliable that specifies files are to be unprotected (no mirroring). Oracle discourages the use of unprotected files; this example is presented only to further illustrate how the attributes for templates are set.

```
ALTER DISKGROUP dgroup2 ADD TEMPLATE unreliable
     ATTRIBUTES (UNPROTECTED);
```

This statement creates a template that applies the following attributes to files:

Template Name	External Redundancy	Normal Redundancy	High Redundancy	Striping
UNRELIABLE	Unprotected	Unprotected	You cannot specify UNRELIABLE for a high redundancy group.	No

Modifying a Disk Group Template

The ALTER TEMPLATE clause of the ALTER DISKGROUP statement enables you to modify the attribute specifications of an existing system default or user-defined disk group template. Only specified template properties are changed. Unspecified properties retain their current value.

When you modify an existing template, only new files created by the template will reflect the attribute changes. Existing files maintain their attributes.

Modifying a Disk Group Template: Example The following example changes the striping attribute specification of the reliable template for disk group dgroup2.

```
ALTER DISKGROUP dgroup2 ALTER TEMPLATE reliable
     ATTRIBUTES (COARSE);
```

Dropping Templates from a Disk Group

Use the DROP TEMPLATE clause of the ALTER DISKGROUP statement to drop one or more templates from a disk group. You can only drop templates that are user-defined; you cannot drop system default templates.

Dropping Templates from a Disk Group: Example This example drops the previously created unprotected template for dgroup2:

```
ALTER DISKGROUP dgroup2 DROP TEMPLATE unreliable;
```

Managing Disk Group Directories

Every ASM disk group contains a hierarchical directory structure consisting of the fully qualified names of the files in the disk group, along with alias filenames. A fully qualified filename, also called a **system alias**, is always generated automatically by Automatic Storage Management when a file is created.

You can create an additional (more user-friendly) alias for each ASM filename during file creation. You can also create an alias for an existing filename using clauses of the ALTER DISKGROUP statement as described in "Managing Alias

Names for ASM Filenames" on page 12-25. But you must first create a directory structure to support whatever alias file naming convention you choose to use.

This section describes how to use the ALTER DISKGROUP statement to create a directory structure for alias filenames. It also describes how you can rename a directory or drop a directory.

> **See Also:** "About ASM Filenames" on page 12-30 for a discussion of ASM filenames and how they are formed

Creating a New Directory

Use the ADD DIRECTORY clause of the ALTER DISKGROUP statement to create a hierarchical directory structure for alias names for ASM files. Use the slash character (/) to separate components of the directory path. The directory path must start with the disk group name, preceded by a plus sign (+), followed by any subdirectory names of your choice.

The parent directory must exist before attempting to create a subdirectory or alias in that directory.

Creating a New Directory: Example 1 The following statement creates a hierarchical directory for disk group dgroup1, which can contain, for example, the alias name +dgroup1/mydir/control_file1:

```
ALTER DISKGROUP dgroup1 ADD DIRECTORY '+dgroup1/mydir';
```

Creating a New Directory: Example 2 Assume no subdirectory exists under the directory +dgoup1/mydir, then the following statement will fail:

```
ALTER DISKGROUP dgroup1
     ADD DIRECTORY 'dgroup1/mydir/does_not_exist/second_dir;
```

Renaming a Directory

The RENAME DIRECTORY clause of the ALTER DISKGROUP statement enables you to rename a directory. System created directories (those containing system alias names) cannot be renamed.

Renaming a Directory: Example The following statement renames a directory:

```
ALTER DISKGROUP dgroup1 RENAME DIRECTORY '+dgroup1/mydir'
     TO '+dgroup1/yourdir';
```

Dropping a Directory

You can delete a directory using the DROP DIRECTORY clause of the ALTER DISKGROUP statement. You cannot drop a system created directory. You cannot drop a directory containing alias names unless you also specify the FORCE clause.

Dropping a Directory: Example This statement deletes a directory along with its contents:

```
ALTER DISKGROUP dgroup1 DROP DIRECTORY '+dgroup1/yourdir' FORCE;
```

Managing Alias Names for ASM Filenames

After you have created the hierarchical directory structure for alias names, you can create alias names in the disk group. Alias names are intended to provide a more user-friendly means of referring to ASM files, rather than using the fully qualified names (system aliases) that Automatic Storage Management always generates when it creates a file.

As mentioned earlier, these alias names can be created when the file is created in the database, or by adding an alias or renaming existing alias names using the ADD ALIAS or RENAME ALIAS clauses of the ALTER DISKGROUP statement. You delete an alias using the DROP ALIAS clause. You cannot delete or rename a system alias.

The V$ASM_ALIAS view contains a row for every alias name known to the ASM instance. It contains a column, SYSTEM_CREATED, that specifies if the alias is system generated

Adding an Alias Name for an ASM Filename

Use the ADD ALIAS clause of the ALTER DISKGROUP statement to create an alias name for an ASM filename. The alias name must consist of the full directory path and the alias itself.

Adding an Alias Name for an ASM Filename: Example 1 The following statement adds a new alias name for a system generated file name:

```
ALTER DISKGROUP dgroup1 ADD ALIAS '+dgroup1/mydir/second.dbf'
     FOR '+dgroupA/sample/datafile/mytable.342.3';
```

Adding an Alias Name for as ASM Filename: Example 2 This statement illustrates another means of specifying the ASM filename for which the alias is to be created. It uses the numeric form of the ASM filename, which is an abbreviated and derived form of the system generated filename.

```
ALTER DISKGROUP dgroup1 ADD ALIAS '+dgroup1/mydir/second.dbf'
     FOR '+dgroupA.342.3';
```

Renaming an Alias Name for an ASM Filename

Use the RENAME ALIAS clause of the ALTER DISKGROUP statement to rename an alias for an ASM filename. The old and the new alias names must consist of the full directory paths of the alias names.

Renaming an Alias Name for an ASM Filename: Example The following statement renames an alias:

```
ALTER DISKGROUP dgroup1 RENAME ALIAS '+dgroup1/mydir/datafile.dbf'
     TO '+dgroupA/payroll/compensation.dbf';
```

Dropping an Alias Name for an ASM Filename

Use the DROP ALIAS clause of the ALTER DISKGROUP statement to drop an alias for an ASM filename. The alias name must consist of the full directory path and the alias itself. The underlying file to which the alias refers is unchanged.

Dropping an Alias Name for an ASM Filename: Example 1 The following statement deletes an alias:

```
ALTER DISKGROUP dgroup1 DELETE ALIAS '+dgroup1/payroll/compensation.dbf';
```

Dropping an Alias Name for an ASM Filename: Example 2 The following statement will fail because it attempts to delete a system alias. This is not allowed:

```
ALTER DISKGROUP dgroup1
     DELETE ALIAS '+dgroup1/sample/datafile/mytable.342.3';
```

Dropping Files and Associated Aliases from a Disk Group

You can delete ASM files and their associated alias names from a disk group using the DROP FILE clause of the ALTER DISKGROUP statement. You must use a fully qualified filename, a numeric filename, or an alias name when specifying the file that you want to delete.

Some reasons why you might need to delete files include:

- Files created using aliases are not Oracle-managed files. Consequently, they are not automatically deleted.
- A point in time recovery of a database might restore the database to a time before a tablespace was created. The restore does not delete the tablespace, but

there is no reference to the tablespace (or its datafile) in the restored database. You can manually delete the datafile.

Dropping an alias does not drop the underlying file on the file system.

Dropping Files and Associated Aliases from a Disk Group: Example 1

The following statement uses the alias name for the file to delete both the file and the alias:

```
ALTER DISKGROUP dgroup1 DROP FILE '+dgroup1/payroll/compensation.dbf';
```

Dropping Files and Associated Aliases from a Disk Group: Example 2

In this example the system generated filename is used to drop the file and any associated alias:

```
ALTER DISKGROUP dgroup1
     DROP FILE '+dgroupA/sample/datafile/mytable.342.372642';
```

Checking Internal Consistency of Disk Group Metadata

You can check the internal consistency of disk group metadata using the `ALTER DISKGROUP ... CHECK` statement. Checking can be specified for specific files in a disk group, specific disks or all disks in a disk group, or specific failure groups within a disk group. The disk group must be mounted in order to perform these checks.

If any errors are detected, an error message is displayed and details of the errors are written to the alert log. Automatic Storage Management attempts to correct any errors, unless you specify the `NOREPAIR` clause in your `ALTER DISKGROUP ... CHECK` statement.

The following statement checks for consistency in the metadata for all disks in the `dgroup1` disk group:

```
ALTER DISKGROUP dgroup1 CHECK ALL;
```

Dropping Disk Groups

The `DROP DISKGROUP` statement enables you to delete an ASM disk group and optionally, all of its files. You can specify the `INCLUDING CONTENTS` clause if you want any files that may still be contained in the disk group also to be deleted. The default is `EXCLUDING CONTENTS`, which provides syntactic consistency and prevents you from dropping the diskgroup if it has any contents

The ASM instance must be started and the disk group must be mounted with none of the disk group files open, in order for the DROP DISKGROUP statement to succeed. The statement does not return until the disk group has been dropped.

When you drop a disk group, Automatic Storage Management dismounts the disk group and removes the disk group name from the ASM_DISKGROUPS initialization parameter if a server parameter file is being used. If a text initialization parameter file is being used, and the disk group is mentioned in the ASM_DISKGROUPS initialization parameter, then you must remember to remove the disk group name from the ASM_DISKGROUPS initialization parameter before the next time that you shut down and restart the ASM instance.

The following statement deletes dgroup1:

```
DROP DISKGROUP dgroup1;
```

After ensuring that none of the files contained in dgroup1 are open, Automatic Storage Management rewrites the header of each disk in the disk group to remove ASM formatting information. The statement does not specify INCLUDING CONTENTS, so the drop operation will fail if the diskgroup contains any files.

Using Automatic Storage Management in the Database

This section discusses how you use Automatic Storage Management to manage database files for you. When you use Automatic Storage Management, Oracle database files are stored in ASM disk groups. These files are not visible to the operating system or its utilities, but are visible to RMAN and other Oracle supplied tools.

This following topics are contained in this section:

- What Types of Files Does Automatic Storage Management Support?
- About ASM Filenames
- Starting the ASM and Database Instances
- Creating and Referencing ASM Files in the Database
- Creating a Database Using Automatic Storage Management
- Creating Tablespaces Using Automatic Storage Management
- Creating Redo Logs Using Automatic Storage Management
- Creating a Control File Using Automatic Storage Management
- Creating Archive Log Files Using Automatic Storage Management

- Recovery Manager (RMAN) and Automatic Storage Management

What Types of Files Does Automatic Storage Management Support?

Automatic Storage Management supports most file types required by the database. However, most administrative files cannot be stored on a ASM disk group. These include trace files, audit files, alert logs, backup files, export files, tar files, and core files.

Table 12–2 lists file types, indicates if they are supported, and lists the system default template that provides the attributes for file creation. Some of the file types shown in the table are related to specific products or features, and are not discussed in this book.

Table 12–2 File Types Supported by Automatic Storage Management

File Type	Supported	Default Templates
Control files	yes	CONTROLFILE
Datafiles	yes	DATAFILE
Redo log files	yes	ONLINELOG
Archive log files	yes	ARCHIVELOG
Trace files	no	n/a
Temporary files	yes	TEMPFILE
Datafile backup pieces	yes	BACKUPSET
Datafile incremental backup pieces	yes	BACKUPSET
Archive log backup piece	yes	BACKUPSET
Datafile copy	yes	DATAFILE
Persistent initialization parameter file (SPFILE)	yes	PARAMETERFILE
Disaster recovery configurations	yes	DATAGUARDCONFIG
Flashback logs	yes	FLASHBACK
Change tracking file	yes	CHANGETRACKING
Data Pump dumpset	yes	DUMPSET
Auto backup	yes	AUTOBACKUP
Operating system files	no	n/a

See Also: "Managing Disk Group Templates" on page 12-21 for a description of the system default templates

About ASM Filenames

ASM filenames can take several forms. Some of the forms are used to create ASM files, others are used to reference them. The forms that you use to create files are alias names or incomplete file names, that basically point to a disk group wherein files are created and given fully qualified names by Automatic Storage Management.

When Automatic Storage Management creates a fully qualified name, an alert log message is written containing this ASM generated name. You can also find the generated name in database views displaying Oracle file names, such as `V$DATAFILE`, `V$LOGFILE`, and so forth. You can use this name, or an abbreviated form of it, if you later need to reference an ASM file in a SQL statement.

Like other Oracle database filenames, ASM filenames are kept in the control file and the RMAN catalog.

These are the forms of an ASM filename:

- Fully Qualified ASM Filename
- Numeric ASM Filename
- Alias ASM Filenames
- Alias ASM Filename with Template
- Incomplete ASM Filename
- Incomplete ASM Filename with Template

The following table specifies the valid contexts for each form of a filename. Single-file creation is relevant for file specifications. Multiple-file creation is relevant in a `*_DEST` parameter.

Filename Form	Valid Context		
	Reference	Single-File Creation	Multiple-File Creation
Fully qualified filename	Yes	No	No
Numeric filename	Yes	No	No
Alias filename	Yes	Yes	No

Filename Form	Valid Context		
	Reference	Single-File Creation	Multiple-File Creation
Alias with template filename	No	Yes	No
Incomplete filename	No	Yes	Yes
Incomplete filename with template	No	Yes	Yes

Note: Fully qualified and numeric filenames can be used in single-file create if you specify the REUSE flag, as described in "Using ASM Filenames in SQL Statements" on page 12-38.

Fully Qualified ASM Filename

This form of ASM filename can be used for referencing existing ASM files. It is the filename that ASM always automatically generates when an ASM file is created. The fully qualified filename is also referred to as the system alias filename.

A fully qualified filename is derived by Automatic Storage Management and has the following form:

```
+group/dbname/file_type/file_type_tag.file.incarnation
```

Where:

- *+group* is the disk group name.
- *dbname* is the DB_UNIQUE_NAME of the database to which the file belongs.
- *file_type* is the Oracle file type and can be one of the file types shown in the table that follows.
- *tag* is type specific information about the file and can be one of the tags shown in the table that follows.
- *file.incarnation* is the file/incarnation pair, used to ensure uniqueness.

An example of a fully qualified ASM filename is:

```
+dgroup2/sample/controlfile/CF.257.1
```

Table 12–3 Oracle File Types and Automatic Storage Management File Type Tags

Automatic Storage Management *file_type*	Description	Automatic Storage Management *file_type_tag*	Comments
CONTROLFILE	Control files and backup control files	Current Backup	--
DATAFILE	Datafiles and datafile copies	*tsname*	Tablespace into which the file is added
ONLINELOG	Online logs	group_*group#*	--
ARCHIVELOG	Archive logs	thread_*thread#*_seq_*sequence#*	--
TEMPFILE	Tempfiles	*tsname*	Tablespace into which the file is added
BACKUPSET	Datafile and archive log backup pieces; datafile incremental backup pieces	*hasspfile_timestamp*	*hasspfile* can take one of two values: *s* indicates that the backup set includes the spfile; *n* indicates that the backup set does not include the spfile.
PARAMETERFILE	Persistent parameter files	spfile	
DAATAGUARDCONFIG	Data Guard configuration file	*db_unique_name*	Data Guard tries to use the service provider name if it is set. Otherwise the tag defaults to *DRCname*.
FLASHBACK	Flashback logs	log_*log#*	--
CHANGETRACKING	Block change tracking data	ctf	Used during incremental backups
DUMPSET	Data Pump dumpset	user_*obj#_file#*	Dump set files encode the user name, the job number that created the dump set, and the file number as part of the tag.
XTRANSPORT	Datafile convert	*tsname*	--
AUTOBACKUP	Automatic backup files	*hasspfile_timestamp*	*hasspfile* can take one of two values: *s* indicates that the backup set includes the spfile; *n* indicates that the backup set does not include the spfile.

Numeric ASM Filename

The numeric ASM filename can be used for referencing existing files. It is derived from the fully qualified ASM filename and takes the form:

```
+group.file.incarnation
```

Numeric ASM filenames can be used in any interface that requires an existing file name.

An example of a numeric ASM filename is:

```
+dgroup2.257.8675309
```

Alias ASM Filenames

Alias ASM filenames can be used both for referencing existing ASM files and for creating new ASM files. Alias names start with the disk group name, after which you specify a name string of your choosing. Alias filenames are implemented using a hierarchical directory structure, with the slash (/) character separating name components. You must have already created the directory structure, using the ALTER DISKGROUP ... CREATE DIRECTORY statement as explained in "Managing Disk Group Directories" on page 12-23.

Alias ASM filenames are distinguished from fully qualified or numeric names because they do not end in a dotted pair of numbers. It is an error to attempt to create an alias that ends in a dotted pair of numbers. Examples of ASM alias filenames are:

```
+dgroup1/myfiles/control_file1
+dgroup2/mydir/second.dbf
```

Files created using an alias are not considered Oracle-managed files and may require manual deletion in the future if they are no longer needed.

An alias ASM filename is normally used in the `CONTROL_FILES` initialization parameter.

Alias ASM Filename with Template

An alias ASM filename with template is used only for ASM file creation operations. They are of the format:

```
+dgroup(template_name)/alias
```

The creation and maintenance of ASM templates is discussed in "Managing Disk Group Templates" on page 12-21. The system default templates that can be specified and attributes that are assigned from these templates, are shown in the table in that section. You can also specify user-defined templates.

An example of an alias ASM filename with template is:

```
+dgroup1(spfile)/config1
```

Explicitly specifying a template name, as in this example, overrides the system default.

Incomplete ASM Filename

Incomplete ASM filenames are used only for file creation operations and are used for both single and multiple-file creation. They consist only of the disk group name. Automatic Storage Management uses a system default template to determine the ASM file redundancy and striping attributes. The system template that is used is determined by the file type that is being created. For example, if you are creating a datafile for a tablespace, then the `datafile` template is used.

An example of an incomplete ASM filename is:

```
+dgroup1
```

Incomplete ASM Filename with Template

Incomplete ASM filenames with templates are used only for file creation operations and are used for both single and multifile creation. They consist of the disk group name followed by the template name in parentheses. When you explicitly specify a template in a file name, ASM uses the specified template instead of the default template for that file type to determine redundancy and striping attributes for the file.

An example of an incomplete ASM filename with template is:

```
+dgroup1(datafile)
```

Starting the ASM and Database Instances

For you to use the functionality of Automatic Storage Management, a database instance requires that an ASM instance is running and that disk groups are configured. Specifically:

1. Start the ASM Instance.

You start the ASM instance on the same node as the database *before* you start the database instance. Starting an ASM instance is discussed in "Starting Up an ASM Instance" on page 12-7

2. Start the database instance

   Consider the following before you start your database instance:

   - To start a database instance, you must have the `INSTANCE_TYPE` initialization parameter set as follows:

     `INSTANCE_TYPE = RDBMS`

     This the default.

   - Specify an ASM filename for any of the following initialization parameters for which you want Automatic Storage Management to automatically create and manage files (see "Creating ASM Files Using a Default Disk Group Specification" on page 12-36:

     - `DB_CREATE_FILE_DEST`
     - `DB_CREATE_ONLINE_LOG_DEST_n`
     - `DB_RECOVERY_FILE_DEST`
     - `CONTROL_FILES`
     - `LOG_ARCHIVE_DEST_n`
     - `LOG_ARCHIVE_DEST`
     - `STANDBY_ARCHIVE_DEST`

   - Some additional initialization parameter considerations:

     - `LOG_ARCHIVE_FORMAT` is ignored if a disk group is specified for `LOG_ARCHIVE_DEST` (for example, `LOG_ARCHIVE_DEST = +dgroup1`).
     - `DB_BLOCK_SIZE` must be one of the standard block sizes (2K, 4K, 8K, 16K or 32K bytes).
     - `LARGE_POOL_SIZE` must be set to at least 8 MB.

Your database instance is now able to create ASM files. You can keep your database instance open and running when you reconfigure disk groups. When you add or remove disks from a disk group, Automatic Storage Management automatically rebalances file data in the reconfigured disk group to ensure a balanced I/O load, even while the database is running.

Creating and Referencing ASM Files in the Database

ASM files are Oracle-managed files unless you created the file using an alias. Any Oracle-managed file is automatically deleted when it is no longer needed. An ASM file is deleted if the creation fails.

Creating ASM Files Using a Default Disk Group Specification

Using the Oracle-managed files feature for operating system files, you can specify a directory as the default location for the creation of datafiles, tempfiles, redo log files, and control files. Using the Oracle-managed files feature for ASM, you can specify a disk group, in the form of an ASM filename, as the default location for creation of these files, and additional types of files, including archived log files. As for operating system files, the name of the default disk group is stored in an initialization parameter and is used whenever a file specification (for example, `DATAFILE` clause) is not explicitly specified during file creation.

The following initialization parameters accept the multifile creation context form of ASM filenames as a destination:

Initialization Parameter	Description
`DB_CREATE_FILE_DEST`	Specifies the default disk group location in which to create: ■ Datafiles ■ Tempfiles If DB_CREATE_ONLINE _LOG_DEST_n is not specified, then also specifies the default disk group for: ■ Redo log files ■ Control file
`DB_CREATE_ONLINE_LOG_DEST_`*n*	Specifies the default disk group location in which to create: ■ Redo log files ■ Control files

Initialization Parameter	Description
DB_RECOVERY_FILE_DEST	If this parameter is specified and DB_CREATE_ONLINE_LOG_DEST_*n* and CONTROL_FILES are not specified, then specifies a default disk group for a flash recovery area that contains a copy of: ■ Control file ■ Redo log files If no local archive destination is specified, then this parameter implicitly sets LOG_ARCHIVE_DEST_10 to USE_DB_RECOVERY_FILE_DEST.
CONTROL_FILES	Specifies a disk group in which to create control files.

The following initialization parameters accept the multifile creation context form of the ASM filenames and ASM directory names as a destination:

Initialization Parameter	Description
LOG_ARCHIVE_DEST_*n*	Specifies a default disk group or ASM directory as destination for archiving redo log files
LOG_ARCHIVE_DEST	Optional parameter to use to specify a default disk group or ASM directory as destination for archiving redo log files. Use when specifying only one destination.
STANDBY_ARCHIVE_DEST	Relevant only for a standby database in managed recovery mode. It specifies a default disk group or ASM directory that is the location of archive logs arriving from a primary database. Not discussed in this book. See *Oracle Data Guard Concepts and Administration*.

The following example illustrates how an ASM file, in this case a datafile, might be created in a default disk group.

Creating a Datafile Using a Default Disk Group: Example Assume the following initialization parameter setting:

```
DB_CREATE_FILE_DEST = '+dgroup1'
```

The following statement creates tablespace tspace1.

```
CREATE TABLESPACE tspace1;
```

Automatic Storage Management automatically creates and manages its datafile on ASM disks in the disk group `dgroup1`. File extents are stored using the attributes defined by the system default template for a datafile.

Using ASM Filenames in SQL Statements

You can specify ASM filenames in the file specification clause of your SQL statements. If you are creating a file for the first time, then use the creation form of an ASM filename. If the ASM file already exists, then the filename must be in a reference context form and, if trying to re-create the file, the REUSE clause specified. The space will be reused for the new file. This usage might occur when, for example, trying to re-create a control file, as shown in "Creating a Control File Using Automatic Storage Management" on page 12-41.

If a reference context form is used with the REUSE clause, and the file does not exist, the numeric portion of the reference context form is ignored, and a new file is created as if the incomplete filename had been specified.

Partially created files resulting from system errors are automatically deleted.

Using an ASM Filename in a SQL Statement: Example The following is an example of specifying an ASM filename in a SQL statement. In this case, it is used in the file creation context:

```
CREATE TABLESPACE  tspace2 DATAFILE '+dgroup2' SIZE 200M AUTOEXTEND ON;
```

The tablespace `tspace2` is created and is comprised of one datafile of size 200M contained in the disk group `dgroup2`. The datafile is set to auto-extensible with an unlimited maximum size. An AUTOEXTEND clause can be used to override this default.

Creating a Database Using Automatic Storage Management

The recommended method of creating your database is to use the Database Configuration Assistant (DBCA). However, if you choose to create your database manually using the CREATE DATABASE statement, then Automatic Storage Management enables you to create a database and all of its underlying files with a minimum of input from you.

The following is an example of using the CREATE DATABASE statement, where database files are created and managed automatically by Automatic Storage Management.

Creating a Database Using Automatic Storage Management: Example

This example creates a database with the following ASM files:

- A SYSTEM tablespace datafile in disk group dgroup1.
- A SYSAUX tablespace datafile in disk group dgroup1. The tablespace is locally managed with automatic segment-space management.
- A multiplexed online redo log is created with two online log groups, one member of each in dgroup1 and dgroup2 (flash recovery area).
- If automatic undo management mode is enabled, then an undo tablespace datafile in directory dgroup1.
- If no CONTROL_FILES initialization parameter is specified, then two control files, one in drgoup1 and another in dgroup2 (flash recovery area). The control file in dgroup1 is the primary control file.

The following initialization parameter settings are included in the initialization parameter file:

```
DB_CREATE_FILE_DEST = '+dgroup1'
DB_RECOVERY_FILE_DEST = '+dgroup2'
DB_RECOVERY_FILE_DEST_SIZE = 10G
```

The following statement is issued at the SQL prompt:

```
SQL> CREATE DATABASE sample;
```

Creating Tablespaces Using Automatic Storage Management

When Automatic Storage Management creates a datafile for a permanent tablespace (or a tempfile for a temporary tablespace), the datafile is set to auto-extensible with an unlimited maximum size and 100 MB default size. You can use the AUTOEXTEND clause to override this default extensibility and the SIZE clause to override the default size.

Automatic Storage Management applies attributes to the datafile, as specified in the system default template for a datafile as shown in the table in "Managing Disk Group Templates" on page 12-21. You can also create and specify your own template.

Files in a tablespace may be in both ASM files and non-ASM files as a result of the tablespace history. RMAN commands enable non-ASM files to be relocated to a ASM disk group and enable ASM files to be relocated as non-ASM files.

The following are some examples of creating tablespaces using Automatic Storage Management. The examples assume that disk groups have already been configured.

See Also:

- *Oracle Database Backup and Recovery Basics*
- *Oracle Database Backup and Recovery Advanced User's Guide*

Creating a Tablespace Using Automatic Storage Management: Example 1

This example illustrates "out of the box" usage of Automatic Storage Management. You let Automatic Storage Management create and manage the tablespace datafile for you, using Oracle supplied defaults that are adequate for most situations.

Assume the following initialization parameter setting:

```
DB_CREATE_FILE_DEST = '+dgroup2'
```

The following statement creates the tablespace and its datafile:

```
CREATE TABLESPACE tspace2;
```

Creating a Tablespace Using Automatic Storage Management: Example 2

The following statements create a tablespace that uses a user defined template (assume it has been defined) to specify the redundancy and striping attributes of the datafile:

```
SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST = '+dgroup1(my_template)';
SQL> CREATE TABLESPACE tspace3;
```

Creating a Tablespace Using Automatic Storage Management: Example 3

The following statement creates an undo tablespace with a datafile that has an alias name and its attributes are set by the user defined template `my_undo_temp`. It assumes a directory has been created in disk group `dgroup3` to contain the alias name and that the user defined template exists. Because an alias is used to create the datafile, the file is not an Oracle-managed file and will not be automatically deleted when the tablespace is dropped.

```
CREATE UNDO TABLESPACE myundo
     DATAFILE '+dgroup3(my_undo_temp)/myfiles/my_undo_ts' SIZE 200M;
```

The following statement drops the file manually after the tablespace has been dropped:

```
ALTER DISKGROUP dgroup3 DROP FLE '+dgroup3/myfiles/my_undots';
```

Creating Redo Logs Using Automatic Storage Management

Online redo logs can be created in multiple disk groups, either implicitly in the initialization parameter file or explicitly in an `ALTER DATABASE ... ADD LOGFILE` statement. Each online log should have one log member in multiple disk groups. The filenames for log file members are automatically generated.

All partially created redo log files, created as a result of a system error, are automatically deleted.

Adding New Redo Log Files: Example

The following example creates a log file with a member in each of the disk groups `dgroup1` and `dgroup2`.

The following parameter settings are included in the initialization parameter file:

```
DB_CREATE_ONLINE_LOG_DEST_1 = '+dgroup1'
DB_CREATE_ONLINE_LOG_DEST_2 = '+dgroup2'
```

The following statement is issued at the SQL prompt:

```
ALTER DATABASE ADD LOGFILE;
```

Creating a Control File Using Automatic Storage Management

Control files can be explicitly created in multiple disk groups. The filenames for control files are automatically generated. If an attempt to create a control file fails, partially created control files will be automatically be deleted.

There may be times when you need to specify a control file by name. Alias filenames are provided to allow administrators to reference ASM files with human-understandable names. The use of an alias in the specification of the control file during its creation allows the DBA to later refer to the control file with a human-meaningful name. Furthermore, an alias can be specified as a control file name in the `CONTROL_FILES` initialization parameter. Control files created without aliases can be given alias names at a later time. The `ALTER DISKGROUP ... CREATE ALIAS` statement is used for this purpose.

When creating a control file, datafiles and log files stored in an ASM disk group should be given to the `CREATE CONTROLFILE` command using the file reference context form of their ASM filenames. However, the use of the `RESETLOGS` option requires the use of a file creation context form for the specification of the log files.

Creating a Control File Using Automatic Storage Management: Example 1

The following CREATE CONTROLFILE statement is generated by an ALTER DATABASE BACKUP CONTROLFILE TO TRACE command for a database with datafiles and log files created on disk groups dgroup1 and dgroup2:

```
CREATE CONTROLFILE REUSE DATABASE "SAMPLE" NORESETLOGS ARCHIVELOG
    MAXLOGFILES 16
    MAXLOGMEMBERS 2
    MAXDATAFILES 30
    MAXINSTANCES 1
    MAXLOGHISTORY 226
LOGFILE
  GROUP 1 (
    '+DGROUP1/db/onlinelog/group_1.258.3',
    '+DGROUP2/db/onlinelog/group_1.256.3'
  ) SIZE 100M,
  GROUP 2 (
    '+DGROUP1/db/onlinelog/group_2.257.3',
    '+DGROUP2/db/onlinelog/group_2.258.1'
  ) SIZE 100M
DATAFILE
  '+DGROUP1/db/datafile/system.260.3',
  '+DGROUP1/db/datafile/sysaux.259.3'
CHARACTER SET US7ASCII
;
```

Creating a Control File Using Automatic Storage Management: Example 2

This example is a CREATE CONTROLFILE statement for a database with datafiles, but uses a RESETLOGS clause, and thus uses the creation context form for log files:

```
CREATE CONTROLFILE REUSE DATABASE "SAMPLE" RESETLOGS ARCHIVELOG
    MAXLOGFILES 16
    MAXLOGMEMBERS 2
    MAXDATAFILES 30
    MAXINSTANCES 1
    MAXLOGHISTORY 226
LOGFILE
  GROUP 1 (
    '+DGROUP1',
    '+DGROUP2'
  ) SIZE 100M,
  GROUP 2 (
    '+DGROUP1',
    '+DGROUP2'
  ) SIZE 100M
```

```
DATAFILE
  '+DGROUP1/db/datafile/system.260.3',
  '+DGROUP1/db/datafile/sysaux.259.3'
CHARACTER SET US7ASCII
;
```

Creating Archive Log Files Using Automatic Storage Management

Disk groups can be specified as archive log destinations in the LOG_ARCHIVE_DEST and LOG_ARCHIVE_DEST_n initialization parameters. When destinations are specified in this manner, the archive log filename will be unique, even if archived twice. Partially created archive operations resulting from a system error are deleted and do not leave a half written file.

If LOG_ARCHIVE_DEST is set to a disk group name, LOG_ARCHIVE_FORMAT is ignored. Unique filenames for archived logs are automatically created by the Oracle database. If LOG_ARCHIVE_DEST is set to a directory in a disk group, LOG_ARCHIVE_FORMAT has its normal semantics.

The following sample archive log names might be generated with DB_RECOVERY_FILE_DEST set to +dgroup2. SAMPLE is the value of the DB_UNIQUE_NAME parameter:

```
+DGROUP2/SAMPLE/ARCHIVELOG/2003_09_23/thread_1_seq_38.614.3
+DGROUP2/SAMPLE/ARCHIVELOG/2003_09_23/thread_4_seq_35.609.3
+DGROUP2/SAMPLE/ARCHIVELOG/2003_09_23/thread_2_seq_34.603.3
+DGROUP2/SAMPLE/ARCHIVELOG/2003_09_25/thread_3_seq_100.621.3
+DGROUP2/SAMPLE/ARCHIVELOG/2003_09_25/thread_1_seq_38.614.3
```

Recovery Manager (RMAN) and Automatic Storage Management

RMAN is critical to Automatic Storage Management and is responsible for tracking the ASM filenames and for deleting obsolete ASM files. Since ASM files cannot be accessed through normal operating system interfaces, RMAN is the preferred means of copying ASM files; you can also use FTP through XDB. RMAN is the only method for performing backups of a database containing ASM files.

RMAN can also be used for moving databases or files into ASM storage.

See Also:

- *Oracle Database Backup and Recovery Basics*
- *Oracle Database Backup and Recovery Advanced User's Guide*

Viewing Information About Automatic Storage Management

You can use these views to query information about Automatic Storage Management:

View	Description
V$ASM_DISKGROUP	In an ASM instance, describes a disk group (number, name, size related info, state, and redundancy type). In a DB instance, contains one row for every ASM disk group mounted by the local ASM instance.
V$ASM_CLIENT	In an ASM instance, identifies databases using disk groups managed by the ASM instance. In a DB instance, contains one row for the ASM instance if the database has any open ASM files.
V$ASM_DISK	In an ASM instance, contains one row for every disk discovered by the ASM instance, including disks that are not part of any disk group. In a DB instance, contains rows only for disks in the disk groups in use by that DB instance.
V$ASM_FILE	In an ASM instance, contains one row for every ASM file in every disk group mounted by the ASM instance. In a DB instance, contains no rows.
V$ASM_TEMPLATE	In an ASM instance, contains one row for every template present in every disk group mounted by the ASM instance. In a DB instance, contains no rows
V$ASM_ALIAS	In an ASM instance, contains one row for every alias present in every disk group mounted by the ASM instance. In a DB instance, contains no rows.
V$ASM_OPERATION	In an ASM instance, contains one row for every active ASM long running operation executing in the ASM instance. In a DB instance, contains no rows.

See Also: *Oracle Database Reference* for details on all of these dynamic performance views

Part IV

Schema Objects

Part IV describes the creation and maintenance of schema objects in the Oracle Database. It includes the following chapters:

13

Managing Space for Schema Objects

This chapter offers guidelines for managing space for schema objects. You should familiarize yourself with the concepts in this chapter before attempting to manage specific schema objects as described in later chapters.

This chapter contains the following topics:

Managing Space in Data Blocks

This section describes aspects of managing space in data blocks. Data blocks are the finest level of granularity of the structure in which database data, including schema object segments, is stored on disk. The size of a data block is determined at database creation by initialization parameters, as discussed in "Specifying Database Block Sizes" on page 2-31.

The PCTFREE and PCTUSED parameters are physical attributes that can be specified when a schema object is created or altered. These parameters allow you to control the use of the free space within a data block. This free space is available for inserts and updates of rows of data.

The PCTFREE and PCTUSED parameters allow you to:

- Improve performance when writing and retrieving data
- Decrease the amount of unused space in data blocks
- Decrease the amount of row migration between data blocks

The INITRANS parameter is another physical attribute that can be specified when schema objects are created or altered. It is used to ensure that a minimum number of concurrent transactions can update the data block.

The following topics are contained in this section:

- Specifying the PCTFREE Parameter
- Specifying the PCTUSED Parameter
- Selecting Associated PCTUSED and PCTFREE Values
- Specifying the INITRANS Parameter

See Also:

- *Oracle Database Concepts* for more information on data blocks
- *Oracle Database SQL Reference* for syntax and other details of the PCTFREE, PCTUSED, and INITRANS physical attributes parameters

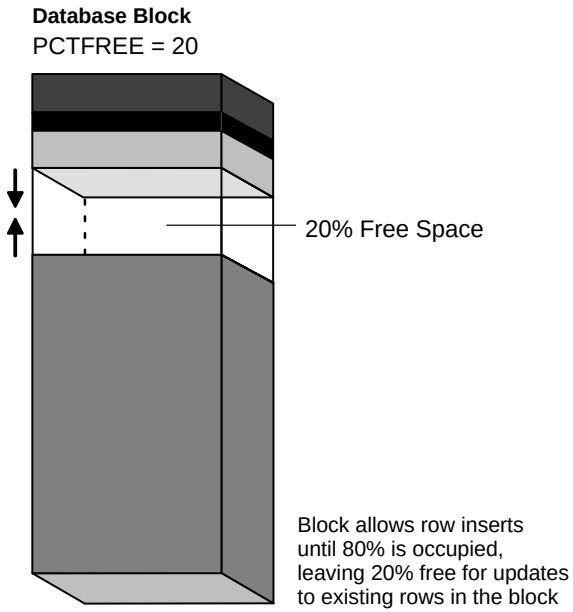
Specifying the PCTFREE Parameter

The PCTFREE parameter is used to set the percentage of a block to be reserved for possible updates to rows that already are contained in that block. For example, assume that you specify the following parameter within a CREATE TABLE statement:

```
PCTFREE 20
```

This indicates that 20% of each data block used for the data segment of this table will be kept free and available for possible updates to the existing rows already within each block. Figure 13–1 illustrates PCTFREE.

Figure 13–1 PCTFREE



Notice that before the block reaches PCTFREE, the free space of the data block is filled by both the insertion of new rows and by the growth of the data block header.

Ensure that you understand the nature of table or index data before setting PCTFREE. Updates can cause rows to grow. New values might not be the same size as values they replace. If there are many updates in which data values get larger, PCTFREE should be increased. If updates to rows do not affect the total row width, PCTFREE can be low. Your goal is to find a satisfactory trade-off between densely packed data and good update performance.

The default for PCTFREE is 10 percent. You can use any integer between 0 and 99, inclusive, as long as the sum of PCTFREE and PCTUSED does not exceed 100.

Effects of Specifying a Smaller PCTFREE

A smaller PCTFREE has the following effects:

- Reserves less room for updates to expand existing table rows
- Allows inserts to fill the block more completely

- May save space, because the total data for a table or index is stored in fewer blocks (more rows or entries for each block)

A small PCTFREE might be suitable, for example, for a segment that is rarely changed.

Effects of Specifying a Larger PCTFREE

A larger PCTFREE has the following effects:

- Reserves more room for future updates to existing table rows
- May require more blocks for the same amount of inserted data (inserting fewer rows for each block)
- May improve update performance, because the database does not need to migrate rows as frequently, if ever

A large PCTFREE is suitable, for example, for segments that are frequently updated.

PCTFREE for Nonclustered Tables

If the data in the rows of a nonclustered table is likely to increase in size over time, reserve some space for these updates. Otherwise, updated rows are likely to be migrated. This happens when updates to a row cause the row to no longer fit in the data block because there is not enough free space left. When a row is migrated, a row piece is left in the original data block pointing to the actual row data stored in another block. This decreases I/O performance.

> **See Also:** "Listing Chained Rows of Tables and Clusters" on page 20-5

PCTFREE for Clustered Tables

The discussion for nonclustered tables also applies to clustered tables. However, if PCTFREE is reached, new rows from *any* table contained in the same cluster key go into a new data block that is chained to the existing cluster key.

PCTFREE for Indexes

You can specify PCTFREE only when initially creating an index.

Specifying the PCTUSED Parameter

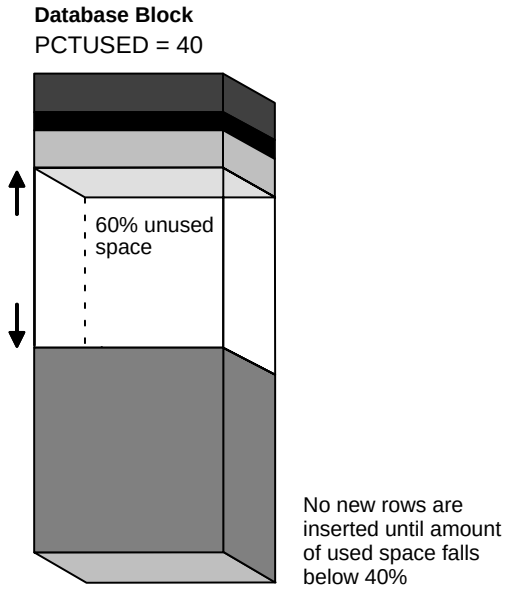
> **Note:** The PCTUSED parameter is ignored for objects created in locally managed tablespaces with segment space management specified as AUTO. This form of segment space management is discussed in "Specifying Segment Space Management in Locally Managed Tablespaces" on page 8-7.

After a data block becomes full as determined by PCTFREE, the database does not consider the block for the insertion of new rows until the percentage of the block being used falls below the parameter PCTUSED. Before this value is achieved, the database uses the free space of the data block only for updates to rows already contained in the data block. For example, assume that you specify the following parameter within a CREATE TABLE statement:

```
PCTUSED 40
```

In this case, a data block used for the data segment of this table is not considered for the insertion of any new rows until the amount of used space in the block falls to 39% or less (assuming that the used space in the block has previously reached PCTFREE). Figure 13–2 illustrates this.

Figure 13–2 PCTUSED



The default value for PCTUSED is 40 percent. After the free space in a data block reaches PCTFREE, no new rows are inserted in that block until the percentage of space used falls below PCTUSED. The percent value is for the block space available for data after overhead is subtracted from total space.

You can specify any integer between 0 and 99 (inclusive) for PCTUSED, as long as the sum of PCTUSED and PCTFREE does not exceed 100.

Effects of Specifying a Smaller PCTUSED

A smaller PCTUSED reduces processing costs incurred during UPDATE and DELETE statements for moving a block to the free list when it has fallen below that percentage of usage. It also increases the unused space in a database.

Effects of Specifying a Larger PCTUSED

A larger PCTUSED improves space efficiency and increases processing cost during INSERT and UPDATE.

Selecting Associated PCTUSED and PCTFREE Values

If you decide not to use the default values for PCTFREE or PCTUSED, keep the following guidelines in mind:

- The sum of PCTFREE and PCTUSED must be equal to or less than 100.
- If the sum equals 100, then the database attempts to keep no more than PCTFREE free space, and processing costs are highest.
- The smaller the difference between 100 and the sum of PCTFREE and PCTUSED (as in PCTUSED of 75, PCTFREE of 20), the more efficient space usage is, at some performance cost.

The following table contains examples that show how and why specific values for PCTFREE and PCTUSED are specified for tables.

Example	Scenario	Settings	Explanation
1	Common activity includes UPDATE statements that increase the size of the rows.	PCTFREE=20 PCTUSED=40	PCTFREE is set to 20 to allow enough room for rows that increase in size as a result of updates. PCTUSED is set to 40 so that less processing is done during high update activity, thus improving performance.
2	Most activity includes INSERT and DELETE statements, and UPDATE statements that do not increase the size of affected rows.	PCTFREE=5 PCTUSED=60	PCTFREE is set to 5 because most UPDATE statements do not increase row sizes. PCTUSED is set to 60 so that space freed by DELETE statements is used soon, yet processing is minimized.
3	The table is very large and storage is a primary concern. Most activity includes read-only transactions.	PCTFREE=5 PCTUSED=40	PCTFREE is set to 5 because this is a large table and you want to completely fill each block.

Specifying the INITRANS Parameter

INITRANS specifies the number of update transaction entries for which space is initially reserved in the data block header. Space is reserved in the headers of all data blocks in the associated segment.

As multiple transactions concurrently access the rows of the same data block, space is allocated for each update transaction entry in the block. Once the space reserved by INITRANS is depleted, space for additional transaction entries is allocated out of the free space in a block, if available. Once allocated, this space effectively becomes a permanent part of the block header.

Note: In earlier releases of Oracle Database, the MAXTRANS parameter limited the number of transaction entries that could concurrently use data in a data block. This parameter has been deprecated. Oracle Database now automatically allows up to 255 concurrent update transactions for any data block, depending on the available space in the block.

The database ignores MAXTRANS when specified by users only for new objects created when the COMPATIBLE initialization parameter is set to 10.0 or greater.

You should consider the following when setting the INITRANS parameter for a schema object:

- The space you would like to reserve for transaction entries compared to the space you would reserve for database data
- The number of concurrent transactions that are likely to touch the same data blocks at any given time

For example, if a table is very large and only a small number of users simultaneously access the table, the chances of multiple concurrent transactions requiring access to the same data block is low. Therefore, INITRANS can be set low, especially if space is at a premium in the database.

Alternatively, assume that a table is usually accessed by many users at the same time. In this case, you might consider preallocating transaction entry space by using a high INITRANS. This eliminates the overhead of having to allocate transaction entry space, as required when the object is in use.

In general, Oracle recommends that you not change the value of INITRANS from its default.

Managing Space in Tablespaces

Oracle Database provides proactive help in managing tablespace disk space use by alerting you when tablespaces run low on available space. Two alert thresholds are defined by default: **warning** and **critical**. The warning threshold is the limit at which space is beginning to run low. The critical threshold is a serious limit that warrants your immediate attention. The database issues alerts at both thresholds.

When you create a new database, all tablespaces are assigned a default of 85% full for the warning threshold and 97% full for the critical threshold. In a migrated database, the thresholds are both set to `NULL`. The `NULL` setting effectively disables the alert mechanism to avoid excessive alerts in a newly migrated database. In either case, you can reset the threshold values, for the entire database or for individual tablespaces, using `DBMS_SERVER_ALERT` package. You use the procedures `SET_THRESHOLD` and `GET_THRESHOLD` to set and get threshold values, respectively.

The ideal setting for the warning threshold is one that issues an alert early enough for you to resolve the problem before it becomes critical. The critical threshold should be one that issues an alert still early enough that you can take immediate action to avoid loss of service.

Threshold-based alerts have the following limitations:

- The default threshold-based alerts and the `DBMS_SERVER_ALERT` package are supported only for locally managed tablespaces. You cannot set alerts for dictionary-managed tablespaces. However, you can monitor dictionary-managed tablespaces using the Enterprise Manager alert system, as you could in earlier releases.
- When you take a tablespace offline or put it in read-only mode, you should disable the alerts for the tablespace by setting the thresholds to `NULL`. You can then reenable the alerts by resetting the thresholds when the tablespace is once again online and in read / write mode.

See Also:

- "Server-Generated Alerts" on page 4-25 for additional information on server-generated alerts in general
- *PL/SQL Packages and Types Reference* for information on the procedures of the DBMS_SERVER_ALERT package and how to use them
- *Oracle Database Performance Tuning Guide* for information on using the Automatic Workload Repository to gather statistics on space usage
- "Reclaiming Unused Space" on page 13-28 for various ways to reclaim space that is no longer being used in the tablespace
- "Purging Objects in the Recycle Bin" on page 14-38 for information on reclaiming recycle bin space

Managing Storage Parameters

This section describes the storage parameters that you can specify for schema object segments to tell the database how to store the object in the database. Schema objects include tables, indexes, partitions, clusters, materialized views, and materialized view logs

The following topics are contained in this section:

- Identifying the Storage Parameters
- Setting Default Storage Parameters for Objects Created in a Tablespace
- Specifying Storage Parameters at Object Creation
- Setting Storage Parameters for Clusters
- Setting Storage Parameters for Partitioned Tables
- Setting Storage Parameters for Index Segments
- Setting Storage Parameters for LOBs, Varrays, and Nested Tables
- Changing Values of Storage Parameters
- Understanding Precedence in Storage Parameters
- Example of How Storage Parameters Effect Space Allocation

Identifying the Storage Parameters

Storage parameters determine space allocation for objects when their segments are created in a tablespace. Not all storage parameters can be specified for every type of database object, and not all storage parameters can be specified in both the CREATE and ALTER statements.

Space allocation is less complex in locally managed tablespaces than in dictionary-managed tablespaces. Storage parameters for objects in locally managed tablespaces are supported mainly for backward compatibility, and they are interpreted differently or are ignored.

For locally managed tablespaces, the Oracle Database server manages extents for you. If you specified the UNIFORM clause when the tablespace was created, then you told the database to create all extents of a uniform size that you specified (or a default size) for any objects created in the tablespace. If you specified the AUTOALLOCATE clause, then the database determines the extent sizing policy for the tablespace. So, for example, if you specific the INITIAL clause when you create an object in a locally managed tablespace you are telling the database to preallocate at least that much space. The database then determines the appropriate number of extents needed to allocate that much space.

Table 13–1 contains a brief description of each storage parameter. For a complete description of these parameters, including their default, minimum, and maximum settings, see the *Oracle Database SQL Reference*.

Table 13–1 Storage Parameters

Parameter	Description
INITIAL	In a dictionary-managed tablespace, the size, in bytes, of the first extent allocated when a segment is created. This parameter cannot be specified in an ALTER statement. In a tablespace that is specified as EXTENT MANAGEMENT LOCAL. The database uses the value of INITIAL in conjunction with the extent size for the tablespace to determine the initial amount of space to reserve for the object. For example, in a uniform locally managed tablespace with 5M extents, if you specify an INITIAL value of 1M, then the database must allocate one 5M extent, because that is the uniform size of extents for the tablespace. If the extent size of the tablespace is smaller than the value of INITIAL, then the initial amount of space allocated will in fact be more than one extent.

Table 13–1 (Cont.) Storage Parameters

Parameter	Description
NEXT	In a dictionary-managed tablespace, the size, in bytes, of the next incremental extent to be allocated for a segment. The second extent is equal to the original setting for NEXT. From there forward, NEXT is set to the previous size of NEXT multiplied by (1 + PCTINCREASE / 100). Not meaningful for objects created in a tablespace that is specified as EXTENT MANAGEMENT LOCAL because the database automatically manages extents.
PCTINCREASE	In a dictionary-managed tablespace, the percentage by which each incremental extent grows over the last incremental extent allocated for a segment. If PCTINCREASE is 0, then all incremental extents are the same size. If PCTINCREASE is greater than zero, then each time NEXT is calculated, it grows by PCTINCREASE. PCTINCREASE cannot be negative. The new NEXT equals 1 + PCTINCREASE / 100, multiplied by the size of the last incremental extent (the old NEXT) and rounded up to the next multiple of a block size. Not meaningful for objects created in a tablespace that is specified as EXTENT MANAGEMENT LOCAL because the database automatically manages the extents.
MINEXTENTS	In a dictionary-managed tablespace, the total number of extents to be allocated when the segment is created. This allows for a large allocation of space at creation time, even if contiguous space is not available. In a tablespace that is specified as EXTENT MANAGEMENT LOCAL, MINEXTENTS is used only to compute the initial amount of space that is allocated. The initial amount of space that is allocated and is equal to INITIAL * MINEXTENTS. Thereafter it is set to 1 for these tablespaces. (as seen in the DBA_SEGMENTS view).
MAXEXTENTS	In a dictionary-managed tablespace, the total number of extents, including the first, that can ever be allocated for the segment. Not meaningful for objects created in locally managed tablespaces because the database automatically manages the extents.
FREELIST GROUPS	The number of groups of free lists for the database object you are creating. The database uses the instance number of Oracle Real Application Cluster instances to map each instance to one free list group. This parameter is ignored for objects created in locally managed tablespaces with segment space management specified as AUTO.

Table 13–1 (Cont.) Storage Parameters

Parameter	Description
FREELISTS	Specifies the number of free lists for each of the free list groups for the schema object. Not valid for tablespaces. This parameter is ignored for objects created in locally managed tablespaces with segment space management specified as AUTO.
BUFFER POOL	Defines a default buffer pool (cache) for a schema object. For information on the use of this parameter, see *Oracle Database Performance Tuning Guide*.

Setting Default Storage Parameters for Objects Created in a Tablespace

When you create a dictionary-managed tablespace you can specify default storage parameters. These values override the system defaults to become the defaults for objects created in that tablespace only. You specify the default storage values in the DEFAULT STORAGE clause of a CREATE or ALTER TABLESPACE statement. For a tablespace which you specifically create as locally managed, the DEFAULT STORAGE clause is not allowed.

Specifying Storage Parameters at Object Creation

At object creation, you can specify storage parameters for each individual schema object. These parameter settings override any default storage settings. Use the STORAGE clause of the CREATE or ALTER statement for specifying storage parameters for the individual object. The following example illustrates specifying storage parameters when a table is being created in a dictionary-managed tablespace:

```
CREATE TABLE  players
        (code  NUMBER(10) PRIMARY KEY,
         lastname  VARCHAR(20),
         firstname VARCHAR(15),
         position  VARCHAR2(20),
         team VARCHAR2(20))
      PCTFREE 10
      PCTUSED 40
      STORAGE
         (INITIAL 25K
          NEXT 10K
          MAXEXTENTS 10
          MINEXTENTS 3);
```

For an object created in a locally managed tablespace, only the INITIAL parameter has meaning.

Setting Storage Parameters for Clusters

You set the storage parameters for nonclustered tables using the STORAGE clause of the CREATE TABLE or ALTER TABLE statement.

In contrast, you set the storage parameters for the data segments of a cluster using the STORAGE clause of the CREATE CLUSTER or ALTER CLUSTER statement, rather than the individual CREATE or ALTER statements that put tables into the cluster. Storage parameters specified when creating or altering a *clustered* table are ignored. The storage parameters set for the cluster override the table storage parameters.

Setting Storage Parameters for Partitioned Tables

With partitioned tables, you can set default storage parameters at the table level. When creating a new partition of the table, the default storage parameters are inherited from the table level (unless you specify them for the individual partition). If no storage parameters are specified at the table level, then they are inherited from the tablespace.

Setting Storage Parameters for Index Segments

Storage parameters for an index segment created for a table index can be set using the STORAGE clause of the CREATE INDEX or ALTER INDEX statement.

Storage parameters of an index segment created for the index used to enforce a primary key or unique key constraint can be set in either of the following ways:

- In the ENABLE ... USING INDEX clause of the CREATE TABLE or ALTER TABLE statement
- In the STORAGE clause of the ALTER INDEX statement

Setting Storage Parameters for LOBs, Varrays, and Nested Tables

A table or materialized view can contain LOB, varray, or nested table column types. These entities can be stored in their own segments. LOBs and varrays are stored in LOB segments, while a nested table is stored in a storage table. You can specify a STORAGE clause for these segments that will override storage parameters specified at the table level.

See Also:

- *Oracle Database Application Developer's Guide - Large Objects* for more information about LOBs
- *Oracle Database Application Developer's Guide - Object-Relational Features* for more information about varrays and nested tables

Changing Values of Storage Parameters

You can alter default storage parameters for tablespaces and specific storage parameters for individual objects if you so choose. Default storage parameters can be reset for a tablespace; however, changes affect only new objects created in the tablespace or new extents allocated for a segment. As discussed previously, you cannot specify default storage parameters for locally managed tablespaces, so this discussion does not apply.

The `INITIAL` and `MINEXTENTS` storage parameters cannot be altered for an existing table, cluster, index. If only `NEXT` is altered for a segment, the next incremental extent is the size of the new `NEXT`, and subsequent extents can grow by `PCTINCREASE` as usual.

If both `NEXT` and `PCTINCREASE` are altered for a segment, the next extent is the new value of `NEXT`, and from that point forward, `NEXT` is calculated using `PCTINCREASE` as usual.

Understanding Precedence in Storage Parameters

Starting with default values, the storage parameters in effect for a database object at a given time are determined by the following, listed in order of precedence (where higher numbers take precedence over lower numbers):

1. Oracle Database default values
2. `DEFAULT STORAGE` clause of `CREATE TABLESPACE` statement
3. `DEFAULT STORAGE` clause of `ALTER TABLESPACE` statement
4. `STORAGE` clause of `CREATE [TABLE | CLUSTER | MATERIALIZED VIEW | MATERIALIZED VIEW LOG | INDEX]` statement
5. `STORAGE` clause of `ALTER [TABLE | CLUSTER | MATERIALIZED VIEW | MATERIALIZED VIEW LOG | INDEX]` statement

Any storage parameter specified at the object level overrides the corresponding option set at the tablespace level. When storage parameters are not explicitly set at

the object level, they default to those at the tablespace level. When storage parameters are not set at the tablespace level, Oracle Database system defaults apply. If storage parameters are altered, the new options apply only to the extents not yet allocated.

> **Note:** The storage parameters for temporary segments always use the default storage parameters set for the associated tablespace.

Example of How Storage Parameters Effect Space Allocation

This discussion applies only to objects created in dictionary-managed tablespaces. For objects created in locally managed tablespaces, extents will either be system managed, or will all be of the same size.

Assume the following statement has been executed:

```
CREATE TABLE test_storage
   ( . . . )
   STORAGE (INITIAL 100K   NEXT 100K
      MINEXTENTS 2   MAXEXTENTS 5
      PCTINCREASE 50);
```

Also assume that the initialization parameter DB_BLOCK_SIZE is set to 2K. The following table shows how extents are allocated for the test_storage table. Also shown is the value for the incremental extent, as can be seen in the NEXT column of the USER_SEGMENTS or DBA_SEGMENTS data dictionary views:

Table 13–2 Extent Allocations

Extent#	Extent Size	Value for NEXT
1	50 blocks or 102400 bytes	50 blocks or 102400 bytes
2	50 blocks or 102400 bytes	75 blocks or153600 bytes
3	75 blocks or 153600 bytes	113 blocks or 231424 bytes
4	115 blocks or 235520 bytes	170 blocks or 348160 bytes
5	170 blocks or 348160 bytes	No next value, MAXEXTENTS=5

If you change the NEXT or PCTINCREASE storage parameters with an ALTER statement (such as ALTER TABLE), the specified value replaces the current value stored in the data dictionary. For example, the following statement modifies the

NEXT storage parameter of the test_storage table before the third extent is allocated for the table:

```
ALTER TABLE test_storage STORAGE (NEXT 500K);
```

As a result, the third extent is 500K when allocated, the fourth is (500K*1.5)=750K, and so forth.

Managing Resumable Space Allocation

Oracle Database provides a means for suspending, and later resuming, the execution of large database operations in the event of space allocation failures. This enables you to take corrective action instead of the Oracle Database server returning an error to the user. After the error condition is corrected, the suspended operation automatically resumes. This feature is called **resumable space allocation**. The statements that are affected are called resumable statements.

This section contains the following topics:

- Resumable Space Allocation Overview
- Enabling and Disabling Resumable Space Allocation
- Detecting Suspended Statements
- Operation-Suspended Alert
- Resumable Space Allocation Example: Registering an AFTER SUSPEND Trigger

Resumable Space Allocation Overview

This section provides an overview of resumable space allocation. It describes how resumable space allocation works, and specifically defines qualifying statements and error conditions.

How Resumable Space Allocation Works

The following is an overview of how resumable space allocation works. Details are contained in later sections.

1. A statement executes in a resumable mode only if its session has been enabled for resumable space allocation by one of the following actions:
   - The RESUMABLE_TIMEOUT initialization parameter is set to a nonzero value.

- The `ALTER SESSION ENABLE RESUMABLE` statement is issued.

2. A resumable statement is suspended when one of the following conditions occur (these conditions result in corresponding errors being signalled for nonresumable statements):
   - Out of space condition
   - Maximum extents reached condition
   - Space quota exceeded condition.
3. When the execution of a resumable statement is suspended, there are mechanisms to perform user supplied operations, log errors, and to query the status of the statement execution. When a resumable statement is suspended the following actions are taken:
   - The error is reported in the alert log.
   - The system issues the Resumable Session Suspended alert.
   - If the user registered a trigger on the `AFTER SUSPEND` system event, the user trigger is executed. A user supplied PL/SQL procedure can access the error message data using the `DBMS_RESUMABLE` package and the `DBA_` or `USER_RESUMABLE` view.
4. Suspending a statement automatically results in suspending the transaction. Thus all transactional resources are held through a statement suspend and resume.
5. When the error condition is resolved (for example, as a result of user intervention or perhaps sort space released by other queries), the suspended statement automatically resumes execution and the Resumable Session Suspended alert is cleared.
6. A suspended statement can be forced to throw the exception using the `DBMS_RESUMABLE.ABORT()` procedure. This procedure can be called by a DBA, or by the user who issued the statement.
7. A suspension time out interval is associated with resumable statements. A resumable statement that is suspended for the timeout interval (the default is two hours) wakes up and returns the exception to the user.
8. A resumable statement can be suspended and resumed multiple times during execution.

What Operations are Resumable?

> **Note:** Resumable space allocation is fully supported when using locally managed tablespaces. There are certain limitations when using dictionary-managed tablespaces. See "Resumable Space Allocation Limitations for Dictionary-Managed Tablespaces" on page 13-20 for details.

The following operations are resumable:

- Queries

  `SELECT` statements that run out of temporary space (for sort areas) are candidates for resumable execution. When using OCI, the calls `OCIStmtExecute()` and `OCIStmtFetch()` are candidates.

- DML

  `INSERT`, `UPDATE`, and `DELETE` statements are candidates. The interface used to execute them does not matter; it can be OCI, SQLJ, PL/SQL, or another interface. Also, `INSERT INTO ... SELECT` from external tables can be resumable.

- Import/Export

  As for SQL*Loader, a command line parameter controls whether statements are resumable after recoverable errors.

- DDL

  The following statements are candidates for resumable execution:

  - `CREATE TABLE ... AS SELECT`
  - `CREATE INDEX`
  - `ALTER INDEX ... REBUILD`
  - `ALTER TABLE ... MOVE PARTITION`
  - `ALTER TABLE ... SPLIT PARTITION`
  - `ALTER INDEX ... REBUILD PARTITION`
  - `ALTER INDEX ... SPLIT PARTITION`
  - `CREATE MATERIALIZED VIEW`

– CREATE MATERIALIZED VIEW LOG

What Errors are Correctable?

There are three classes of correctable errors:

- Out of space condition

  The operation cannot acquire any more extents for a table / index / temporary segment / undo segment / cluster / LOB / table partition / index partition in a tablespace. For example, the following errors fall in this category:

  ```
  ORA-1653 unable to extend table ... in tablespace ...
  ORA-1654 unable to extend index ... in tablespace ...
  ```

- Maximum extents reached condition

  The number of extents in a table / index / temporary segment / undo segment / cluster / LOB / table partition / index partition equals the maximum extents defined on the object. For example, the following errors fall in this category:

  ```
  ORA-1631 max # extents ... reached in table ...
  ORA-1654 max # extents ... reached in index ...
  ```

- Space quota exceeded condition

  The user has exceeded his assigned space quota in the tablespace. Specifically, this is noted by the following error:

  ```
  ORA-1536 space quote exceeded for tablespace string
  ```

Resumable Space Allocation Limitations for Dictionary-Managed Tablespaces

There are certain limitations of resumable space allocation when using dictionary-managed tablespaces:

1. If a DDL operation such as CREATE TABLE or CREATE INDEX is executed with an explicit MAXEXTENTS setting (or uses the MAXEXTENTS setting from the tablespace DEFAULT STORAGE clause) which causes an out of space error during its execution, the operation will not be suspended. Instead, it will be aborted. This error is treated as not repairable because the properties of an object (for example, MAXEXTENTS) cannot be altered before its creation. However if a DML operation causes an already existing table or index to reach the MAXEXTENTS limit, it will be suspended and can be resumed later. This restriction can be overcome either by setting the MAXEXTENTS clause to UNLIMITED or by using locally managed tablespaces.

2. If rollback segments are located in dictionary-managed tablespaces, then space allocation for rollback segments is not resumable. However, space allocation for user objects (tables, indexes, and the likes) would still be resumable. To work around the limitation, Oracle recommends using automatic undo management or placing the rollback segments in locally managed tablespaces.

Resumable Space Allocation and Distributed Operations

In a distributed environment, if a user enables or disables resumable space allocation, or if you, as a DBA, alter the RESUMABLE_TIMEOUT initialization parameter, only the local instance is affected. In a distributed transaction, sessions or remote instances are suspended only if RESUMABLE has been enabled in the remote instance.

Parallel Execution and Resumable Space Allocation

In parallel execution, if one of the parallel execution server processes encounters a correctable error, that server process suspends its execution. Other parallel execution server processes will continue executing their respective tasks, until either they encounter an error or are blocked (directly or indirectly) by the suspended server process. When the correctable error is resolved, the suspended process resumes execution and the parallel operation continues execution. If the suspended operation is terminated, the parallel operation aborts, throwing the error to the user.

Different parallel execution server processes may encounter one or more correctable errors. This may result in firing an AFTER SUSPEND trigger multiple times, in parallel. Also, if a parallel execution server process encounters a noncorrectable error while another parallel execution server process is suspended, the suspended statement is immediately aborted.

For parallel execution, every parallel execution coordinator and server process has its own entry in the DBA_ or USER_RESUMABLE view.

Enabling and Disabling Resumable Space Allocation

Resumable space allocation is only possible when statements are executed within a session that has resumable mode enabled. There are two means of enabling and disabling resumable space allocation. You can control it at the system level with the RESUMABLE_TIMEOUT initialization parameter, or users can enable it at the session level using clauses of the ALTER SESSION statement.

Note: Because suspended statements can hold up some system resources, users must be granted the RESUMABLE system privilege before they are allowed to enable resumable space allocation and execute resumable statements.

Setting the RESUMABLE_TIMEOUT Initialization Parameter

You can enable resumable space allocation system wide and specify a timeout interval by setting the RESUMABLE_TIMEOUT initialization parameter. For example, the following setting of the RESUMABLE_TIMEOUT parameter in the initialization parameter file causes all sessions to initially be enabled for resumable space allocation and sets the timeout period to 1 hour:

```
RESUMABLE_TIMEOUT  = 3600
```

If this parameter is set to 0, then resumable space allocation is disabled initially for all sessions. This is the default.

You can use the ALTER SYSTEM SET statement to change the value of this parameter at the system level. For example, the following statement will disable resumable space allocation for all sessions:

```
ALTER SYSTEM SET RESUMABLE_TIMEOUT=0;
```

Within a session, a user can issue the ALTER SESSION SET statement to set the RESUMABLE_TIMEOUT initialization parameter and enable resumable space allocation, change a timeout value, or to disable resumable mode.

Using ALTER SESSION to Enable and Disable Resumable Space Allocation

A user can enable resumable mode for a session, using the following SQL statement:

```
ALTER SESSION ENABLE RESUMABLE;
```

To disable resumable mode, a user issues the following statement:

```
ALTER SESSION DISABLE RESUMABLE;
```

The default for a new session is resumable mode disabled, unless the RESUMABLE_TIMEOUT initialization parameter is set to a nonzero value.

The user can also specify a timeout interval, and can provide a name used to identify a resumable statement. These are discussed separately in following sections.

> **See Also:** "Using a LOGON Trigger to Set Default Resumable Mode" on page 13-23

Specifying a Timeout Interval A timeout period, after which a suspended statement will error if no intervention has taken place, can be specified when resumable mode is enabled. The following statement specifies that resumable transactions will time out and error after 3600 seconds:

```
ALTER SESSION ENABLE RESUMABLE TIMEOUT 3600;
```

The value of TIMEOUT remains in effect until it is changed by another ALTER SESSION ENABLE RESUMABLE statement, it is changed by another means, or the session ends. The default timeout interval when using the ENABLE RESUMABLE TIMEOUT clause to enable resumable mode is 7200 seconds.

> **See Also:** "Setting the RESUMABLE_TIMEOUT Initialization Parameter" on page 13-22 for other methods of changing the timeout interval for resumable space allocation

Naming Resumable Statements Resumable statements can be identified by name. The following statement assigns a name to resumable statements:

```
ALTER SESSION ENABLE RESUMABLE TIMEOUT 3600 NAME 'insert into table';
```

The NAME value remains in effect until it is changed by another ALTER SESSION ENABLE RESUMABLE statement, or the session ends. The default value for NAME is 'User *username*(*userid*), Session *sessionid*, Instance *instanceid*'.

The name of the statement is used to identify the resumable statement in the DBA_RESUMABLE and USER_RESUMABLE views.

Using a LOGON Trigger to Set Default Resumable Mode

Another method of setting default resumable mode, other than setting the RESUMABLE_TIMEOUT initialization parameter, is that you can register a database level LOGON trigger to alter a user's session to enable resumable and set a timeout interval.

Note: If there are multiple triggers registered that change default mode and timeout for resumable statements, the result will be unspecified because Oracle Database does not guarantee the order of trigger invocation.

Detecting Suspended Statements

When a resumable statement is suspended, the error is not raised to the client. In order for corrective action to be taken, Oracle Database provides alternative methods for notifying users of the error and for providing information about the circumstances.

Notifying Users: The AFTER SUSPEND System Event and Trigger

When a resumable statement encounter a correctable error, the system internally generates the AFTER SUSPEND system event. Users can register triggers for this event at both the database and schema level. If a user registers a trigger to handle this system event, the trigger is executed after a SQL statement has been suspended.

SQL statements executed within a AFTER SUSPEND trigger are always nonresumable and are always autonomous. Transactions started within the trigger use the SYSTEM rollback segment. These conditions are imposed to overcome deadlocks and reduce the chance of the trigger experiencing the same error condition as the statement.

Users can use the USER_RESUMABLE or DBA_RESUMABLE views, or the DBMS_RESUMABLE.SPACE_ERROR_INFO function, within triggers to get information about the resumable statements.

Triggers can also call the DBMS_RESUMABLE package to terminate suspended statements and modify resumable timeout values. In the following example, the default system timeout is changed by creating a system wide AFTER SUSPEND trigger that calls DBMS_RESUMABLE to set the timeout to 3 hours:

```
CREATE OR REPLACE TRIGGER resumable_default_timeout
AFTER SUSPEND
ON DATABASE
BEGIN
   DBMS_RESUMABLE.SET_TIMEOUT(10800);
END;
```

> **See Also:** *Oracle Database Application Developer's Guide - Fundamentals* for information about system events, triggers, and attribute functions

Using Views to Obtain Information About Suspended Statements

The following views can be queried to obtain information about the status of resumable statements:

View	Description
DBA_RESUMABLE USER_RESUMABLE	These views contain rows for all currently executing or suspended resumable statements. They can be used by a DBA, AFTER SUSPEND trigger, or another session to monitor the progress of, or obtain specific information about, resumable statements.
V$SESSION_WAIT	When a statement is suspended the session invoking the statement is put into a wait state. A row is inserted into this view for the session with the EVENT column containing "statement suspended, wait error to be cleared".

See Also: *Oracle Database Reference* for specific information about the columns contained in these views

Using the DBMS_RESUMABLE Package

The DBMS_RESUMABLE package helps control resumable space allocation. The following procedures can be invoked:

Procedure	Description
ABORT(sessionID)	This procedure aborts a suspended resumable statement. The parameter sessionID is the session ID in which the statement is executing. For parallel DML/DDL, sessionID is any session ID which participates in the parallel DML/DDL. Oracle Database guarantees that the ABORT operation always succeeds. It may be called either inside or outside of the AFTER SUSPEND trigger. The caller of ABORT must be the owner of the session with sessionID, have ALTER SYSTEM privilege, or have DBA privileges.
GET_SESSION_ TIMEOUT(sessionID)	This function returns the current timeout value of resumable space allocation for the session with sessionID. This returned timeout is in seconds. If the session does not exist, this function returns -1.
SET_SESSION_ TIMEOUT(sessionID, timeout)	This procedure sets the timeout interval of resumable space allocation for the session with sessionID. The parameter timeout is in seconds. The new timeout setting will applies to the session immediately. If the session does not exist, no action is taken.

Procedure	Description
GET_TIMEOUT()	This function returns the current timeout value of resumable space allocation for the current session. The returned value is in seconds.
SET_ TIMEOUT(timeout)	This procedure sets a timeout value for resumable space allocation for the current session. The parameter timeout is in seconds. The new timeout setting applies to the session immediately.

See Also: *PL/SQL Packages and Types Reference*

Operation-Suspended Alert

When a resumable session is suspended, an operation-suspended alert is issued on the object that needs allocation of resource for the operation to complete. Once the resource is allocated and the operation completes, the operation-suspended alert is cleared. Please refer to "Managing Space in Tablespaces" on page 13-9 for more information on system-generated alerts.

Resumable Space Allocation Example: Registering an AFTER SUSPEND Trigger

In the following example, a system wide AFTER SUSPEND trigger is created and registered as user SYS at the database level. Whenever a resumable statement is suspended in any session, this trigger can have either of two effects:

- If an undo segment has reached its space limit, then a message is sent to the DBA and the statement is aborted.
- If any other recoverable error has occurred, the timeout interval is reset to 8 hours.

Here are the statements for this example:

```
CREATE OR REPLACE TRIGGER resumable_default
AFTER SUSPEND
ON DATABASE
DECLARE
   /* declare transaction in this trigger is autonomous */
   /* this is not required because transactions within a trigger
      are always autonomous */
   PRAGMA AUTONOMOUS_TRANSACTION;
   cur_sid           NUMBER;
   cur_inst          NUMBER;
```

```
   errno             NUMBER;
   err_type          VARCHAR2;
   object_owner      VARCHAR2;
   object_type       VARCHAR2;
   table_space_name  VARCHAR2;
   object_name       VARCHAR2;
   sub_object_name   VARCHAR2;
   error_txt         VARCHAR2;
   msg_body          VARCHAR2;
   ret_value         BOOLEAN;
   mail_conn         UTL_SMTP.CONNECTION;
BEGIN
   -- Get session ID
   SELECT DISTINCT(SID) INTO cur_SID FROM V$MYSTAT;

   -- Get instance number
   cur_inst := userenv('instance');

   -- Get space error information
   ret_value :=
   DBMS_RESUMABLE.SPACE_ERROR_INFO(err_type,object_type,object_owner,
        table_space_name,object_name, sub_object_name);
   /*
   -- If the error is related to undo segments, log error, send email
   -- to DBA, and abort the statement. Otherwise, set timeout to 8 hours.
   --
   -- sys.rbs_error is a table which is to be
   -- created by a DBA manually and defined as
   -- (sql_text VARCHAR2(1000), error_msg VARCHAR2(4000),
   -- suspend_time DATE)
   */

   IF OBJECT_TYPE = 'UNDO SEGMENT' THEN
       /* LOG ERROR */
       INSERT INTO sys.rbs_error (
           SELECT SQL_TEXT, ERROR_MSG, SUSPEND_TIME
           FROM DBMS_RESUMABLE
           WHERE SESSION_ID = cur_sid AND INSTANCE_ID = cur_inst
        );
       SELECT ERROR_MSG INTO error_txt FROM DBMS_RESUMABLE
           WHERE SESSION_ID = cur_sid and INSTANCE_ID = cur_inst;

        -- Send email to receipient via UTL_SMTP package
        msg_body:='Subject: Space Error Occurred
```

```
                    Space limit reached for undo segment ' || object_name ||
                    on ' || TO_CHAR(SYSDATE, 'Month dd, YYYY, HH:MIam') ||
                    '. Error message was ' || error_txt;

        mail_conn := UTL_SMTP.OPEN_CONNECTION('localhost', 25);
        UTL_SMTP.HELO(mail_conn, 'localhost');
        UTL_SMTP.MAIL(mail_conn, 'sender@localhost');
        UTL_SMTP.RCPT(mail_conn, 'recipient@localhost');
        UTL_SMTP.DATA(mail_conn, msg_body);
        UTL_SMTP.QUIT(mail_conn);

        -- Abort the statement
        DBMS_RESUMABLE.ABORT(cur_sid);
    ELSE
        -- Set timeout to 8 hours
        DBMS_RESUMABLE.SET_TIMEOUT(28800);
    END IF;

    /* commit autonomous transaction */
    COMMIT;
END;
```

Reclaiming Unused Space

Over time, it is common for segment space to become fragmented or for a segment to acquire a lot of free space as the result of update and delete operations. The resulting sparsely populated objects can suffer performance degradation during queries and DML operations.

The Segment Advisor can help you determine which objects have space available for reclamation. If the Segment Advisor indicates that an object does have space available for reclamation, you have two ways to release the space so that it can be used by other segments: you can compact and shrink database segments or you can deallocate unused space at the end of a database segment.

The Segment Advisor relies for its analysis on data collected in the Automatic Workload Repository (AWR). An adjustment to the collection interval and retention period for AWR statistics can affect the precision and the type of recommendations the advisor produces. Please refer to "Automatic Workload Repository" on page 1-26 for more information.

Segment Advisor

Oracle Database provides a Segment Advisor that helps you determine whether an object has space available for reclamation based on the level of space fragmentation within the object. The Segment Advisor can generate advice at three levels:

- At the **object level**, advice is generated for the entire object, such as a table. If the object is partitioned, then the advice is generated on all the partitions of the object. However, advice does not cascade to dependent objects such as indexes, LOB segments, and so forth.
- At the **segment level**, the advice is generated for a single segment, such as unpartitioned table, a partition or subpartition of a partitioned table, or an index or LOB column.
- At the **tablespace level**, advice is generated for every segment in the tablespace.

The best way to invoke the Segment Advisor is through Enterprise Manager. Please refer to *Oracle 2 Day DBA* for more information on how to use Enterprise Manager to invoke the Segment Advisor. In addition, you activate the Segment Advisor using procedures of the `DBMS_ADVISOR` package. Please refer to *PL/SQL Packages and Types Reference* for details on these parameters. The following procedures are relevant for the Segment Advisor:

CREATE_TASK Use this procedure to create the Segment Advisor Task. Specify 'Segment Advisor' as the value of the `ADVISOR_NAME` parameter.

CREATE_OBJECT Use this procedure to identify the target object for segment space advice. The parameter values of this procedure depend upon the object type. Table 13–3 lists the parameter values for each type of object.

Table 13–3 Input for DBMS_ADVISOR.CREATE_OBJECT

Input Parameter				
OBJECT_TYPE	**ATTR1**	**ATTR2**	**ATTR3**	**ATTR4**
`TABLESPACE`	*`tablespace_name`*	`NULL`	`NULL`	Unused. Specify `NULL`.
`TABLE`	*`schema_name`*	*`table_name`*	`NULL`	Unused. Specify `NULL`.
`INDEX`	*`schema_name`*	*`index_name`*	`NULL`	Unused. Specify `NULL`.
`TABLE PARTITION`	*`schema_name`*	*`table_name`*	*`table_partition_name`*	Unused. Specify `NULL`.

Table 13–3 (Cont.) Input for DBMS_ADVISOR.CREATE_OBJECT

Input Parameter				
OBJECT_TYPE	**ATTR1**	**ATTR2**	**ATTR3**	**ATTR4**
INDEX PARTITION	*schema_name*	*index_name*	*index_partition_name*	Unused. Specify NULL.
TABLE SUBPARTITION	*schema_name*	*table_name*	*table_subpartition_name*	Unused. Specify NULL.
INDEX SUBPARTITION	*schema_name*	*index_name*	*index_subpartition_name*	Unused. Specify NULL.

SET_TASK_PARAMETER Use this procedure to describe the segment advice you need. Table 13–4 shows the relevant input parameters of this procedure. Parameters not listed here are not used by the Segment Advisor.

Table 13–4 Input for DBMS_ADVISOR.SET_TASK_PARAMETER

Input Parameter	Description	Possible Values	Default Value
MODE	The data to use for analyzing the segment.	LIMITED: Analysis is restricted to statistics available in the Automatic Workload Repository. COMPREHENSIVE: Analysis is based on both sampling and Automatic Workload Repository statistics.	COMPREHENSIVE
TIME_LIMIT	The time limit for which the advisor should run, specified in seconds.	Any number of seconds	UNLIMITED
RECOMMEND_ALL	Whether the advisor should generate a recommendation for all segments.	TRUE: Findings are generated on all segments specified. FALSE: Findings are generated only for those objects that are eligible for shrink segment.	TRUE

The example that follows shows how to use the DBMS_ADVISOR procedures to activate the Segment Advisor for the sample table hr.employees. The user executing these procedures must have the EXECUTE object privilege on the package or the ADVISOR system privilege:

```
variable id number;
```

```
begin
  declare
  name varchar2(100);
  descr varchar2(500);
  obj_id number;
  begin
  name:='';
  descr:='Segment Advisor Example';
  dbms_advisor.create_task('Segment Advisor', :id, name, descr, NULL);
  dbms_advisor.create_object
     (name, 'TABLE', 'HR', 'EMPLOYEES', NULL, NULL, obj_id);
 dbms_advisor.set_task_parameter(name, 'RECOMMEND_ALL', 'TRUE');
 dbms_advisor.execute_task(name);
 end;
 end;
 /
```

The Segment Advisor creates several types of results:

Findings If you have specified TRUE for RECOMMEND_ALL in the SET_TASK_PARAMETER procedure, then the advisor generates a finding for each segment that qualifies for analysis. You can retrieve the findings by querying the DBA_ADVISOR_FINDINGS data dictionary view.

Recommendations If segment shrink would result in benefit, then the advisor generates a recommendation for the segment. You can retrieve the recommendations by querying the DBA_ADVISOR_RECOMMENDATIONS dictionary view.

Actions Every advisory recommendation is associated with a suggested action to perform segment shrink. You can retrieve the action by querying the DBA_ADVISOR_ACTIONS data dictionary review. This view provides the SQL you need to perform segment shrink.

Objects All findings, recommendations, and actions are associated with an object. If the input to the advisor results in analysis of more than one segment, as with a tablespace or partitioned table, then one entry is created for each segment in the DBA_ADVISOR_OBJECTS dictionary view. You can correlate the object in this view with the object in the findings, recommendations, and actions views.

Please refer to *Oracle Database Reference* for a description of these views.

Table 13–5 shows which dictionary columns contain output from the Segment Advisor and summarizes the possible advisor outcomes.

Table 13–5 Segment Advisor Outcomes: Summary

MESSAGE column of DBA_ADVISOR_ FINDINGS	MORE_INFO column of DBA_ADVISOR_ FINDINGS	BENEFIT_TYPE column of DBA_ADVISOR_ RECOMMENDATIONS	ATTR1 column of DBA_ ADVISOR_ACTIONS
Insufficient information to make a recommendation	None	None	None
Object has less than 1% free space	Displays the amount of allocated, used, and reclaimable space.	None	None
Object has free space but its properties are not compatible with segment shrink.	Displays the amount of allocated, used, and reclaimable space.	None	None
Free space in the object is less than the size of the last extent.	Displays the amount of allocated, used, and reclaimable space.	None	None
Shrink operation is recommended and the estimated savings in bytes is displayed.	Displays the amount of allocated, used, and reclaimable space.	Shrink operation is recommended and the estimated savings in bytes is displayed.	`ALTER` *`object`* `SHRINK SPACE;`

Shrinking Database Segments

Oracle Database lets you shrink space in a table, index-organized table, index, partition, subpartition, materialized view, or materialized view log. You do this using `ALTER TABLE`, `ALTER INDEX`, `ALTER MATERIALIZED VIEW`, or `ALTER MATERIALIZED VIEW LOG` statement with the `SHRINK SPACE` clause. Shrink operations can be performed only on segments in tablespaces with automatic segment-space management. As with other DDL operations, segment shrink causes subsequent SQL statements to be reparsed because of invalidation of cursors unless you specify the `COMPACT` clause, described below. Segment shrink is available through the Enterprise Manager interface, which guides you through the required steps. The remainder of this section discusses manually implementing segment shrink.

Segment shrink reclaims unused space both above and below the high water mark. In contrast, space deallocation reclaims unused space only above the high water mark. In shrink operations, by default, the database compacts the segment, adjusts the high water mark, and releases the reclaimed space. Two optional clauses let you control how the shrink operation proceeds:

- The COMPACT clause lets you divide the shrink segment operation into two phases. When you specify COMPACT, Oracle Database defragments the segment space and compacts the table rows but postpones the resetting of the high water mark and the reallocation of the space until a future time. This option is useful if you have long-running queries that might span the operation and attempt to read from blocks that have been reclaimed. The defragmentation and compaction results are saved to disk, so the data movement does not have to be redone during the second phase. You can reissue the SHRINK SPACE clause without the COMPACT clause during off-peak hours to complete the second phase.
- The CASCADE clause extends the segment shrink operation to all dependent segments of the object. For example, if you specify CASCADE when shrinking a table segment, all indexes of the table will also be shrunk. (You need not specify CASCADE to shrink the partitions of a partitioned table.) To see a list of dependent segments of a given object, you can run the OBJECT_DEPENDENT_SEGMENTS procedure of the DBMS_SPACE package.

Segment shrink is an online, in-place operation. Unlike other space reorganization methods, segment shrink does not require extra disk space to be allocated. Indexes are maintained during the shrink operation and remain usable after the operation is complete. DML operations and queries can be issued during the data movement phase of segment shrink. Concurrent DML operation are blocked for a short time at the end of the shrink operation, when the space is deallocated.

Additional benefits of shrink operations are these:

- Compaction of data leads to better cache utilization, which in turn leads to better online transaction processing (OLTP) performance.
- The compacted data requires fewer blocks to be scanned in full table scans, which in turns leads to better decision support system (DSS) performance.

Segment shrink requires that rows be moved to new locations. Therefore, you must first enable row movement in the object you want to shrink and disable any rowid-based triggers defined on the object. Segment shrink is not supported for LOB segments or for tables with function-based indexes. Please refer to *Oracle Database SQL Reference* for the syntax and additional information on shrinking a database object, including restrictions.

Deallocating Unused Space

When you deallocate unused space, the database frees the unused space at the unused (high water mark) end of the database segment and makes the space available for other segments in the tablespace.

Prior to deallocation, you can run the UNUSED_SPACE procedure of the DBMS_SPACE package, which returns information about the position of the high water mark and the amount of unused space in a segment. For segments in locally managed tablespaces with automatic segment-space management, use the SPACE_USAGE procedure for more accurate information on unused space.

> **See Also:** *PL/SQL Packages and Types Reference* contains the description of the DBMS_SPACE package

The following statements deallocate unused space in a segment (table, index or cluster):

```
ALTER TABLE table DEALLOCATE UNUSED KEEP integer;
ALTER INDEX index DEALLOCATE UNUSED KEEP integer;
ALTER CLUSTER cluster DEALLOCATE UNUSED KEEP integer;
```

The KEEP clause is optional and lets you specify the amount of space retained in the segment. You can verify the deallocated space is freed by examining the DBA_FREE_SPACE view.

> **See Also:**
>
> - *Oracle Database SQL Reference* for details on the syntax and semantics of deallocating unused space
> - *Oracle Database Reference* for more information about the DBA_FREE_SPACE view

Understanding Space Usage of Datatypes

When creating tables and other data structures, you need to know how much space they will require. Each datatype has different space requirements. The *PL/SQL User's Guide and Reference* and *Oracle Database SQL Reference* contain extensive descriptions of datatypes and their space requirements.

Displaying Information About Space Usage for Schema Objects

Oracle Database provides data dictionary views and PL/SQL packages that allow you to display information about the space usage of schema objects. Views and packages that are unique to a particular schema object are described in the chapter of this book associated with that object. This section describes views and packages that are generic in nature and apply to multiple schema objects.

Using PL/SQL Packages to Display Information About Schema Object Space Usage

These Oracle-supplied PL/SQL packages provide information about schema objects:

Package and Procedure/Function	Description
DBMS_SPACE.UNUSED_SPACE	Returns information about unused space in an object (table, index, or cluster).
DBMS_SPACE.FREE_BLOCKS	Returns information about free data blocks in an object (table, index, or cluster) whose segment free space is managed by free lists (segment space management is MANUAL).
DBMS_SPACE.SPACE_USAGE	Returns information about free data blocks in an object (table, index, or cluster) whose segment space management is AUTO.

See Also: *PL/SQL Packages and Types Reference* for a description of PL/SQL packages

Example: Using DBMS_SPACE.UNUSED_SPACE

The following SQL*Plus example uses the DBMS_SPACE package to obtain unused space information.

```
SQL> VARIABLE total_blocks NUMBER
SQL> VARIABLE total_bytes NUMBER
SQL> VARIABLE unused_blocks NUMBER
SQL> VARIABLE unused_bytes NUMBER
SQL> VARIABLE lastextf NUMBER
SQL> VARIABLE last_extb NUMBER
SQL> VARIABLE lastusedblock NUMBER
SQL> exec DBMS_SPACE.UNUSED_SPACE('SCOTT', 'EMP', 'TABLE', :total_blocks, -
>    :total_bytes,:unused_blocks, :unused_bytes, :lastextf, -
>    :last_extb, :lastusedblock);
```

```
PL/SQL procedure successfully completed.

SQL> PRINT

TOTAL_BLOCKS
------------
           5

TOTAL_BYTES
-----------
      10240

...

LASTUSEDBLOCK
-------------
            3
```

Using Views to Display Information About Space Usage in Schema Objects

These views display information about space usage in schema objects:

View	Description
DBA_SEGMENTS USER_SEGMENTS	DBA view describes storage allocated for all database segments. User view describes storage allocated for segments for the current user.
DBA_EXTENTS USER_EXTENTS	DBA view describes extents comprising all segments in the database. User view describes extents comprising segments for the current user.
DBA_FREE_SPACE USER_FREE_SPACE	DBA view lists free extents in all tablespaces. User view shows free space information for tablespaces for which the user has quota.

The following sections contain examples of using some of these views.

> **See Also:** *Oracle Database Reference* for a complete description of data dictionary views

Example 1: Displaying Segment Information

The following query returns the name and size of each index segment in schema `hr`:

```
SELECT SEGMENT_NAME, TABLESPACE_NAME, BYTES, BLOCKS, EXTENTS
    FROM DBA_SEGMENTS
    WHERE SEGMENT_TYPE = 'INDEX'
    AND OWNER='HR'
    ORDER BY SEGMENT_NAME;
```

The query output is:

```
SEGMENT_NAME              TABLESPACE_NAME    BYTES BLOCKS EXTENTS
------------------------- --------------- -------- ------ -------
COUNTRY_C_ID_PK           EXAMPLE            65536     32       1
DEPT_ID_PK                EXAMPLE            65536     32       1
DEPT_LOCATION_IX          EXAMPLE            65536     32       1
EMP_DEPARTMENT_IX         EXAMPLE            65536     32       1
EMP_EMAIL_UK              EXAMPLE            65536     32       1
EMP_EMP_ID_PK             EXAMPLE            65536     32       1
EMP_JOB_IX                EXAMPLE            65536     32       1
EMP_MANAGER_IX            EXAMPLE            65536     32       1
EMP_NAME_IX               EXAMPLE            65536     32       1
JHIST_DEPARTMENT_IX       EXAMPLE            65536     32       1
JHIST_EMPLOYEE_IX         EXAMPLE            65536     32       1
JHIST_EMP_ID_ST_DATE_PK   EXAMPLE            65536     32       1
JHIST_JOB_IX              EXAMPLE            65536     32       1
JOB_ID_PK                 EXAMPLE            65536     32       1
LOC_CITY_IX               EXAMPLE            65536     32       1
LOC_COUNTRY_IX            EXAMPLE            65536     32       1
LOC_ID_PK                 EXAMPLE            65536     32       1
LOC_STATE_PROVINCE_IX     EXAMPLE            65536     32       1
REG_ID_PK                 EXAMPLE            65536     32       1

19 rows selected.
```

Example 2: Displaying Extent Information

Information about the currently allocated extents in a database is stored in the DBA_EXTENTS data dictionary view. For example, the following query identifies the extents allocated to each index segment in the hr schema and the size of each of those extents:

```
SELECT SEGMENT_NAME, SEGMENT_TYPE, TABLESPACE_NAME, EXTENT_ID, BYTES, BLOCKS
    FROM DBA_EXTENTS
    WHERE SEGMENT_TYPE = 'INDEX'
    AND OWNER='HR'
    ORDER BY SEGMENT_NAME;
```

The query output is:

```
SEGMENT_NAME              SEGMENT_TYPE TABLESPACE_NAME EXTENT_ID    BYTES BLOCKS
------------------------- ------------ -------------- --------- -------- ------
COUNTRY_C_ID_PK           INDEX        EXAMPLE                0    65536     32
DEPT_ID_PK                INDEX        EXAMPLE                0    65536     32
DEPT_LOCATION_IX          INDEX        EXAMPLE                0    65536     32
EMP_DEPARTMENT_IX         INDEX        EXAMPLE                0    65536     32
EMP_EMAIL_UK              INDEX        EXAMPLE                0    65536     32
EMP_EMP_ID_PK             INDEX        EXAMPLE                0    65536     32
EMP_JOB_IX                INDEX        EXAMPLE                0    65536     32
EMP_MANAGER_IX            INDEX        EXAMPLE                0    65536     32
EMP_NAME_IX               INDEX        EXAMPLE                0    65536     32
JHIST_DEPARTMENT_IX       INDEX        EXAMPLE                0    65536     32
JHIST_EMPLOYEE_IX         INDEX        EXAMPLE                0    65536     32
JHIST_EMP_ID_ST_DATE_PK   INDEX        EXAMPLE                0    65536     32
JHIST_JOB_IX              INDEX        EXAMPLE                0    65536     32
JOB_ID_PK                 INDEX        EXAMPLE                0    65536     32
LOC_CITY_IX               INDEX        EXAMPLE                0    65536     32
LOC_COUNTRY_IX            INDEX        EXAMPLE                0    65536     32
LOC_ID_PK                 INDEX        EXAMPLE                0    65536     32
LOC_STATE_PROVINCE_IX     INDEX        EXAMPLE                0    65536     32
REG_ID_PK                 INDEX        EXAMPLE                0    65536     32

19 rows selected.
```

For the hr schema, no segment has more than one extent allocated to it.

Example 3: Displaying the Free Space (Extents) in a Tablespace

Information about the free extents (extents not allocated to any segment) in a database is stored in the DBA_FREE_SPACE data dictionary view. For example, the following query reveals the amount of free space available as free extents in the SMUNDO tablespace:

```
SELECT TABLESPACE_NAME, FILE_ID, BYTES, BLOCKS
    FROM DBA_FREE_SPACE
    WHERE TABLESPACE_NAME='SMUNDO';
```

The query output is:

```
TABLESPACE_NAME  FILE_ID    BYTES BLOCKS
--------------- -------- -------- ------
SMUNDO                 3    65536     32
SMUNDO                 3    65536     32
SMUNDO                 3    65536     32
```

```
SMUNDO                  3    65536     32
SMUNDO                  3    65536     32
SMUNDO                  3    65536     32
SMUNDO                  3   131072     64
SMUNDO                  3   131072     64
SMUNDO                  3    65536     32
SMUNDO                  3  3407872   1664

10 rows selected.
```

Example 4: Displaying Segments that Cannot Allocate Additional Extents

It is possible that a segment cannot be allocated to an extent for any of the following reasons:

- The tablespace containing the segment does not have enough room for the next extent.
- The segment has the maximum number of extents.
- The segment has the maximum number of extents allowed by the data block size, which is operating system specific.

The following query returns the names, owners, and tablespaces of all segments that satisfy any of these criteria:

```
SELECT a.SEGMENT_NAME, a.SEGMENT_TYPE, a.TABLESPACE_NAME, a.OWNER
    FROM DBA_SEGMENTS a
    WHERE a.NEXT_EXTENT >= (SELECT MAX(b.BYTES)
        FROM DBA_FREE_SPACE b
        WHERE b.TABLESPACE_NAME = a.TABLESPACE_NAME)
    OR a.EXTENTS = a.MAX_EXTENTS
    OR a.EXTENTS = 'data_block_size' ;
```

> **Note:** When you use this query, replace *data_block_size* with the data block size for your system.

Once you have identified a segment that cannot allocate additional extents, you can solve the problem in either of two ways, depending on its cause:

- If the tablespace is full, add a datafile to the tablespace or extend the existing datafile.
- If the segment has too many extents, and you cannot increase MAXEXTENTS for the segment, perform the following steps.

1. Export the data in the segment
2. Drop and re-create the segment, giving it a larger `INITIAL` storage parameter setting so that it does not need to allocate so many extents. Alternatively, you can adjust the `PCTINCREASE` and `NEXT` storage parameters to allow for more space in the segment.
3. Import the data back into the segment.

Capacity Planning for Database Objects

Oracle Database provides three procedures in the `DBMS_SPACE` package that let you predict the size of new objects and monitor the size of existing database objects. This section discusses those procedures and contains the following sections:

- Estimating the Space Use of a Table
- Estimating the Space Use of an Index
- Obtaining Object Growth Trends

Estimating the Space Use of a Table

The size of a database table can vary greatly depending on tablespace storage attributes, tablespace block size, and many other factors. The `CREATE_TABLE_COST` procedure of the `DBMS_SPACE` package lets you estimate the space use cost of creating a table. Please refer to [ARPLS] for details on the parameters of this procedure.

The procedure has two variants. The first variant uses average row size to estimate size. The second variant uses column information to estimate table size. Both variants require as input the following values:

- `TABLESPACE_NAME`: The tablespace in which the object will be created. The default is the `SYSTEM` tablespace.
- `ROW_COUNT`: The anticipated number of rows in the table.
- `PCT_FREE`: The percentage of free space you want to reserve in each block for future expansion of existing rows due to updates.

In addition, the first variant also requires as input a value for `AVG_ROW_SIZE`, which is the anticipated average row size in bytes.

The second variant also requires for each anticipated column values for COLINFOS, which is an object type comprising the attributes COL_TYPE (the datatype of the column) and COL_SIZE (the number of characters or bytes in the column).

The procedure returns two values:

- USED_BYTES: The actual bytes used by the data, including overhead for block metadata, PCT_FREE space, and so forth.
- ALLOC_BYTES: The amount of space anticipated to be allocated for the object taking into account the tablespace extent characteristics.

Estimating the Space Use of an Index

The CREATE_INDEX_COST procedure of the DBMS_SPACE package lets you estimate the space use cost of creating an index on an existing table.

The procedure requires as input the following values:

- DDL: The CREATE INDEX statement that would create the index. The table specified in this DDL statement must be an existing table.
- [Optional] PLAN_TABLE: The name of the plan table to use. The default is NULL.

The results returned by this procedure depend on statistics gathered on the segment. Therefore, be sure to obtain statistics shortly before executing this procedure. In the absence of recent statistics, the procedure does not issue an error, but it may return inappropriate results. The procedure returns the following values:

- USED_BYTES: The number of bytes representing the actual index data.
- ALLOC_BYTES: The amount of space allocated for the index in the tablespace.

Obtaining Object Growth Trends

The OBJECT_GROWTH_TREND procedure of the DBMS_SPACE package produces a table of one or more rows, where each row describes the space use of the object at a specific time. The procedure retrieves the space use totals from the Automatic Workload Repository or computes current space use and combines it with historic space use changes retrieved from Automatic Workload Repository. Please refer to [ARPLS] for detailed information on the parameters of this procedure.

The procedure requires as input the following values:

- OBJECT_OWNER: The owner of the object.

- OBJECT_NAME: The name of the object.
- PARTITION_NAME: The name of the table or index partition, is relevant. Specify NULL otherwise.
- OBJECT_TYPE: The type of the object.
- START_TIME: A TIMESTAMP value indicating the beginning of the growth trend analysis.
- END_TIME: A TIMESTAMP value indicating the end of the growth trend analysis. The default is "NOW".
- INTERVAL: The length in minutes of the reporting interval during which the procedure should retrieve space use information.
- SKIP_INTERPOLATED: Determines whether the procedure should omit values based on recorded statistics before and after the INTERVAL ('YES') or not ('NO'). This setting is useful when the result table will be displayed as a table rather than a chart, because you can see more clearly how the actual recording interval relates to the requested reporting interval.

The procedure returns a table, each of row of which provides space use information on the object for one interval. If the return table is very large, the results are pipelined so that another application can consume the information as it is being produced. The output table has the following columns:

- TIMEPOINT: A TIMESTAMP value indicating the time of the reporting interval.

  Records are not produced for values of TIME that precede the oldest recorded statistics for the object.
- SPACE_USAGE: The number of bytes actually being used by the object data.
- SPACE_ALLOC: The number of bytes allocated to the object in the tablespace at that time.
- QUALITY: A value indicating how well the requested reporting interval matches the actual recording of statistics. This information is useful because there is no guaranteed reporting interval for object size use statistics, and the actual reporting interval varies over time and from object to object.

  The values of the QUALITY column are:

  – GOOD: The value whenever the value of TIME is based on recorded statistics with a recorded timestamp within 10% of the INTERVAL specified in the input parameters.

- INTERPOLATED: The value did not meet the criteria for GOOD, but was based on recorded statistics before and after the value of TIME. Current in-memory statistics can be collected across all instances in a cluster and treated as the "recorded" value for the present time.
- PROJECTION: The value of TIME is in the future as of the time the table was produced. In a Real Application Clusters environment, the rules for recording statistics allow each instance to choose independently which objects will be selected.

The output returned by this procedure is an aggregation of values recorded across all instances in a RAC environment. Each value can be computed from a combination of GOOD and INTERPOLATED values. The aggregate value returned is marked GOOD if at least 80% of that value was derived from GOOD instance values.

14

Managing Tables

This chapter describes the various aspects of managing tables, and includes the following topics:

About Tables

Tables are the basic unit of data storage in an Oracle Database. Data is stored in rows and columns. You define a table with a table name, such as `employees`, and a set of columns. You give each column a column name, such as `employee_id`,

last_name, and job_id; a datatype, such as VARCHAR2, DATE, or NUMBER; and a width. The width can be predetermined by the datatype, as in DATE. If columns are of the NUMBER datatype, define precision and scale instead of width. A row is a collection of column information corresponding to a single record.

You can specify rules for each column of a table. These rules are called integrity constraints. One example is a NOT NULL integrity constraint. This constraint forces the column to contain a value in every row.

After you create a table, insert rows of data using SQL statements. Table data can then be queried, deleted, or updated using SQL.

See Also:

- *Oracle Database Concepts* for a more detailed description of tables
- Chapter 13, "Managing Space for Schema Objects" is recommended reading before attempting tasks described in this chapter
- Chapter 20, "General Management of Schema Objects" presents additional aspects of managing tables, such as specifying integrity constraints and analyzing tables

Guidelines for Managing Tables

This section describes guidelines to follow when managing tables. Following these guidelines can make the management of your tables easier and can improve performance when creating the table, as well as when loading, updating, and querying the table data.

The following topics are discussed:

- Design Tables Before Creating Them
- Consider Your Options for the Type of Table to Create
- Specify How Data Block Space Is to Be Used
- Specify the Location of Each Table
- Consider Parallelizing Table Creation
- Consider Using NOLOGGING When Creating Tables
- Estimate Table Size and Plan Accordingly

- Restrictions to Consider When Creating Tables

Design Tables Before Creating Them

Usually, the application developer is responsible for designing the elements of an application, including the tables. Database administrators are responsible for establishing the attributes of the underlying tablespace that will hold the application tables. Either the DBA or the applications developer, or both working jointly, can be responsible for the actual creation of the tables, depending upon the practices for a site.

Working with the application developer, consider the following guidelines when designing tables:

- Use descriptive names for tables, columns, indexes, and clusters.
- Be consistent in abbreviations and in the use of singular and plural forms of table names and columns.
- Document the meaning of each table and its columns with the `COMMENT` command.
- Normalize each table.
- Select the appropriate datatype for each column.
- Define columns that allow nulls last, to conserve storage space.
- Cluster tables whenever appropriate, to conserve storage space and optimize performance of SQL statements.

Before creating a table, you should also determine whether to use integrity constraints. Integrity constraints can be defined on the columns of a table to enforce the business rules of your database automatically.

> **See Also:** *Oracle Database Application Developer's Guide - Fundamentals* for information about designing tables

Consider Your Options for the Type of Table to Create

What types of tables can you create? Here are some choices:

Type of Table	Description
Ordinary (heap-organized) table	This is the basic, general purpose type of table which is the primary subject of this chapter. Its data is stored as an unordered collection (heap)
Clustered table	A clustered table is a table that is part of a cluster. A cluster is a group of tables that share the same data blocks because they share common columns and are often used together. Clusters and clustered tables are discussed in Chapter 17, "Managing Clusters".
Index-organized table	Unlike an ordinary (heap-organized) table, data for an index-organized table is stored in a B-tree index structure in a primary key sorted manner. Besides storing the primary key column values of an index-organized table row, each index entry in the B-tree stores the nonkey column values as well. Index-organized tables are discussed in "Managing Index-Organized Tables" on page 14-40.
Partitioned table	Partitioned tables allow your data to be broken down into smaller, more manageable pieces called partitions, or even subpartitions. Each partition can be managed individually, and can operate independently of the other partitions, thus providing a structure that can be better tuned for availability and performance. Partitioned tables are discussed in Chapter 16, "Managing Partitioned Tables and Indexes".

Specify How Data Block Space Is to Be Used

By specifying the PCTFREE and PCTUSED parameters during the creation of each table, you can affect the efficiency of space utilization and amount of space reserved for updates to the current data in the data blocks of a table data segment. The PCTFREE and PCTUSED parameters are discussed in "Managing Space in Data Blocks" on page 13-1.

Note: When you create a table in a locally managed tablespace for which automatic segment-space management is enabled, the need to specify the PCTUSED (or FREELISTS) parameter is eliminated, and if specified, is ignored. Automatic segment-space management is specified at the tablespace level. The Oracle Database server automatically and efficiently manages free and used space within objects created in such tablespaces.

Locally managed tablespaces and automatic segment space management are discussed in "Locally Managed Tablespaces" on page 8-4.

Specify the Location of Each Table

It is advisable to specify the TABLESPACE clause in a CREATE TABLE statement to identify the tablespace that is to store the new table. Ensure that you have the appropriate privileges and quota on any tablespaces that you use. If you do not specify a tablespace in a CREATE TABLE statement, the table is created in your default tablespace.

When specifying the tablespace to contain a new table, ensure that you understand implications of your selection. By properly specifying a tablespace during the creation of each table, you can increase the performance of the database system and decrease the time needed for database administration.

The following situations illustrate how not specifying a tablespace, or specifying an inappropriate one, can affect performance:

- If users' objects are created in the SYSTEM tablespace, the performance of the database can suffer, since both data dictionary objects and user objects must contend for the same datafiles. Users' objects should not be stored in the SYSTEM tablespace. To avoid this, ensure that all users are assigned default tablespaces when they are created in the database.
- If application-associated tables are arbitrarily stored in various tablespaces, the time necessary to complete administrative operations (such as backup and recovery) for the data of that application can be increased.

Consider Parallelizing Table Creation

You can utilize parallel execution when creating tables using a subquery (AS SELECT) in the CREATE TABLE statement. Because multiple processes work together to create the table, performance of the table creation operation is improved.

Parallelizing table creation is discussed in the section "Parallelizing Table Creation" on page 14-10.

Consider Using NOLOGGING When Creating Tables

To create a table most efficiently use the NOLOGGING clause in the CREATE TABLE ... AS SELECT statement. The NOLOGGING clause causes minimal redo information to be generated during the table creation. This has the following benefits:

- Space is saved in the redo log files.
- The time it takes to create the table is decreased.
- Performance improves for parallel creation of large tables.

The NOLOGGING clause also specifies that subsequent direct loads using SQL*Loader and direct load INSERT operations are not logged. Subsequent DML statements (UPDATE, DELETE, and conventional path insert) are unaffected by the NOLOGGING attribute of the table and generate redo.

If you cannot afford to lose the table after you have created it (for example, you will no longer have access to the data used to create the table) you should take a backup immediately after the table is created. In some situations, such as for tables that are created for temporary use, this precaution may not be necessary.

In general, the relative performance improvement of specifying NOLOGGING is greater for larger tables than for smaller tables. For small tables, NOLOGGING has little effect on the time it takes to create a table. However, for larger tables the performance improvement can be significant, especially when you are also parallelizing the table creation.

Consider Using Table Compression when Creating Tables

The Oracle Database **table compression** feature compresses data by eliminating duplicate values in a database block. Duplicate values in all the rows and columns in a block are stored once at the beginning of the block, in what is called a **symbol table** for that block. All occurrences of such values are replaced with a short reference to the symbol table.

Using table compression reduces disk use and memory use in the buffer cache, often resulting in better scale-up for read-only operations. Table compression can also speed up query execution. There is, however, a slight cost in CPU overhead.

Compression occurs when data is inserted with a bulk (direct-path) insert operation. A table can consist of compressed and uncompressed blocks transparently. Any DML operation can be applied to a table storing compressed blocks. However, conventional DML operations cause records to be stored uncompressed afterward the operations and the operations themselves are subject to a small performance overhead due to the nature of the table compression.

Consider using table compression when your data is mostly read only. Do not use table compression for tables that updated frequently.

Estimate Table Size and Plan Accordingly

Estimate the sizes of tables before creating them. Preferably, do this as part of database planning. Knowing the sizes, and uses, for database tables is an important part of database planning.

You can use the combined estimated size of tables, along with estimates for indexes, undo space, and redo log files, to determine the amount of disk space that is required to hold an intended database. From these estimates, you can make correct hardware purchases.

You can use the estimated size and growth rate of an individual table to better determine the attributes of a tablespace and its underlying datafiles that are best suited for the table. This can enable you to more easily manage the table disk space and improve I/O performance of applications that use the table.

Restrictions to Consider When Creating Tables

Here are some restrictions that may affect your table planning and usage:

- Tables containing object types cannot be imported into a pre-Oracle8 database.
- You cannot merge an exported table into a preexisting table having the same name in a different schema.
- You cannot move types and extent tables to a different schema when the original data still exists in the database.
- Oracle Database has a limit on the total number of columns that a table (or attributes that an object type) can have. See *Oracle Database Reference* for this limit.

Further, when you create a table that contains user-defined type data, the database maps columns of user-defined type to relational columns for storing the user-defined type data. This causes additional relational columns to be created. This results in "hidden" relational columns that are not visible in a DESCRIBE table statement and are not returned by a SELECT * statement. Therefore, when you create an object table, or a relational table with columns of REF, varray, nested table, or object type, be aware that the total number of columns that the database actually creates for the table can be more than those you specify.

> **See Also:** *Oracle Database Application Developer's Guide - Object-Relational Features* for more information about user-defined types

Creating Tables

To create a new table in your schema, you must have the CREATE TABLE system privilege. To create a table in another user's schema, you must have the CREATE ANY TABLE system privilege. Additionally, the owner of the table must have a quota for the tablespace that contains the table, or the UNLIMITED TABLESPACE system privilege.

Create tables using the SQL statement CREATE TABLE.

This section contains the following topics:

- Creating a Table
- Creating a Temporary Table
- Parallelizing Table Creation

> **See Also:** *Oracle Database SQL Reference* for exact syntax of the CREATE TABLE and other SQL statements discussed in this chapter

Creating a Table

When you issue the following statement, you create a table named admin_emp in the hr schema and store it in the admin_tbs tablespace with an initial extent size of 50K:

```
CREATE TABLE        hr.admin_emp (
         empno      NUMBER(5) PRIMARY KEY,
         ename      VARCHAR2(15) NOT NULL,
```

```
        job         VARCHAR2(10),
        mgr         NUMBER(5),
        hiredate    DATE DEFAULT (sysdate),
        sal         NUMBER(7,2),
        comm        NUMBER(7,2),
        deptno      NUMBER(3) NOT NULL
                     CONSTRAINT admin_dept_fkey REFERENCES hr.departments
                     (department_id))
   TABLESPACE admin_tbs
   STORAGE ( INITIAL 50K);
```

In this CREATE TABLE statement, integrity constraints are defined on several columns of the table. Integrity constraints are discussed in "Managing Integrity Constraints" on page 20-12.

> **See Also:** *Oracle Database SQL Reference* for description of the datatypes that can be specified for columns

Creating a Temporary Table

It is also possible to create a temporary table. The definition of a temporary table is visible to all sessions, but the data in a temporary table is visible only to the session that inserts the data into the table. Use the CREATE GLOBAL TEMPORARY TABLE statement to create a temporary table. The ON COMMIT clause indicate if the data in the table is **transaction-specific** (the default) or **session-specific**, the implications of which are as follows:

ON COMMIT Setting	Description
DELETE ROWS	This creates a temporary table that is transaction specific. A session becomes bound to the temporary table with a transactions first insert into the table. The binding goes away at the end of the transaction. The database truncates the table (delete all rows) after each commit.
PRESERVE ROWS	This creates a temporary table that is session specific. A session gets bound to the temporary table with the first insert into the table in the session. This binding goes away at the end of the session or by issuing a TRUNCATE of the table in the session. The database truncates the table when you terminate the session.

Temporary tables are useful in applications where a result set is to be buffered, perhaps because it is constructed by running multiple DML operations. For example, consider the following:

A Web-based airlines reservations application allows a customer to create several optional itineraries. Each itinerary is represented by a row in a temporary table. The application updates the rows to reflect changes in the itineraries. When the customer decides which itinerary she wants to use, the application moves the row for that itinerary to a persistent table.

During the session, the itinerary data is private. At the end of the session, the optional itineraries are dropped.

This statement creates a temporary table that is transaction specific:

```
CREATE GLOBAL TEMPORARY TABLE admin_work_area
        (startdate DATE,
         enddate DATE,
         class CHAR(20))
      ON COMMIT DELETE ROWS;
```

Indexes can be created on temporary tables. They are also temporary and the data in the index has the same session or transaction scope as the data in the underlying table.

Unlike permanent tables, temporary tables and their indexes do not automatically allocate a segment when they are created. Instead, segments are allocated when the first INSERT (or CREATE TABLE AS SELECT) is performed. This means that if a SELECT, UPDATE, or DELETE is performed before the first INSERT, the table appears to be empty.

DDL operations (except TRUNCATE) are allowed on an existing temporary table only if no session is currently bound to that temporary table.

If you rollback a transaction, the data you entered is lost, although the table definition persists.

A transaction-specific temporary table allows only one transaction at a time. If there are several autonomous transactions in a single transaction scope, each autonomous transaction can use the table only as soon as the previous one commits.

Because the data in a temporary table is, by definition, temporary, backup and recovery of temporary table data is not available in the event of a system failure. To prepare for such a failure, you should develop alternative methods for preserving temporary table data.

Parallelizing Table Creation

When you specify the AS SELECT clause to create a table and populate it with data from another table, you can utilize parallel execution. The CREATE TABLE ... AS

SELECT statement contains two parts: a CREATE part (DDL) and a SELECT part (query). Oracle Database can parallelize both parts of the statement. The CREATE part is parallelized if *one* of the following is true:

- A PARALLEL clause is included in the CREATE TABLE ... AS SELECT statement
- An ALTER SESSION FORCE PARALLEL DDL statement is specified

The query part is parallelized if *all* of the following are true:

- The query includes a parallel hint specification (PARALLEL or PARALLEL_INDEX) *or* the CREATE part includes the PARALLEL clause *or* the schema objects referred to in the query have a PARALLEL declaration associated with them.
- At least one of the tables specified in the query requires either a full table scan *or* an index range scan spanning multiple partitions.

If you parallelize the creation of a table, that table then has a parallel declaration (the PARALLEL clause) associated with it. Any subsequent DML or queries on the table, for which parallelization is possible, will attempt to use parallel execution.

The following simple statement parallelizes the creation of a table and stores the result in a compressed format, using table compression:

```
CREATE TABLE hr.admin_emp_dept
     PARALLEL COMPRESS
     AS SELECT * FROM hr.employees
     WHERE department_id = 10;
```

In this case, the PARALLEL clause tells the database to select an optimum number of parallel execution servers when creating the table.

See Also:

- *Oracle Database Concepts* for more information about parallel execution
- *Oracle Data Warehousing Guide* for a detailed discussion about using parallel execution
- "Managing Processes for Parallel SQL Execution" on page 4-19

Inserting Data Into Tables Using Direct-Path INSERT

There are several means of inserting or initially loading data into your tables. Most commonly used might be the following:

Method	Description
SQL*Loader	This Oracle utility program loads data from external files into tables of an Oracle Database. For information about SQL*Loader, see *Oracle Database Utilities*.
`CREATE TABLE ... AS SELECT` statement (CTAS)	Using this SQL statement you can create a table and populate it with data selected from another existing table.
`INSERT` statement	The `INSERT` statement enables you to add rows to a table, either by specifying the column values or by selecting data from another existing table.
`MERGE` statement	The `MERGE` statement enables you to insert rows into or update rows of a table, by selecting rows from another existing table. If a row in the new data corresponds to an item that already exists in the table, then an `UPDATE` is performed, else an `INSERT` is performed.

Oracle Database inserts data into a table in one of two ways:

- During **conventional INSERT operations**, the database reuses free space in the table, interleaving newly inserted data with existing data. During such operations, the database also maintains referential integrity constraints.
- During **direct-path INSERT operations**, the database appends the inserted data after existing data in the table. Data is written directly into datafiles, bypassing the buffer cache. Free space in the existing data is not reused, and referential integrity constraints are ignored. These procedures combined can enhance performance.

Further, the data can be inserted either in serial mode, where one process executes the statement, or parallel mode, where multiple processes work together simultaneously to run a single SQL statement. The latter is referred to as parallel execution.

This section discusses one aspect of inserting data into tables. Specifically, using the direct-path form of the `INSERT` statement. It contains the following topics:

- Additional Considerations for Direct-Path INSERT

Note: Only a few details and examples of inserting data into tables are included in this book. Oracle documentation specific to data warehousing and application development provide more extensive information about inserting and manipulating data in tables. For example:

- *Oracle Data Warehousing Guide*
- *Oracle Database Application Developer's Guide - Fundamentals*
- *Oracle Database Application Developer's Guide - Large Objects*

Advantages of Using Direct-Path INSERT

The following are performance benefits of direct-path `INSERT`:

- During direct-path `INSERT`, you can disable the logging of redo and undo entries. Conventional insert operations, in contrast, must always log such entries, because those operations reuse free space and maintain referential integrity.
- To create a new table with data from an existing table, you have the choice of creating the new table and then inserting into it, or executing a `CREATE TABLE ... AS SELECT` statement. By creating the table and then using direct-path `INSERT` operations, you update any indexes defined on the target table during the insert operation. The table resulting from a `CREATE TABLE ... AS SELECT` statement, in contrast, does not have any indexes defined on it; you must define them later.
- Direct-path `INSERT` operations ensure atomicity of the transaction, even when run in parallel mode. Atomicity cannot be guaranteed during parallel direct-path loads (using SQL*Loader).
- If errors occur during parallel direct-path loads, some indexes could be marked `UNUSABLE` at the end of the load. Parallel direct-path `INSERT`, in contrast, rolls back the statement if errors occur during index update.
- Direct-path `INSERT` must be used if you want to store the data in compressed form using table compression.

Enabling Direct-Path INSERT

You can implement direct-path INSERT operations by using direct-path INSERT statements, inserting data in parallel mode, or by using the Oracle SQL*Loader utility in direct-path mode. Direct-path inserts can be done in either serial or parallel mode.

To activate direct-path INSERT in serial mode, you must specify the APPEND hint in each INSERT statement, either immediately after the INSERT keyword, or immediately after the SELECT keyword in the subquery of the INSERT statement.

When you are inserting in parallel DML mode, direct-path INSERT is the default. In order to run in parallel DML mode, the following requirements must be met:

- You must have Oracle Enterprise Edition installed.
- You must enable parallel DML in your session. To do this, run the following statement:

  ```
  ALTER SESSION { ENABLE | FORCE } PARALLEL DML;
  ```

- You must specify the parallel attribute for the target table, either at create time or subsequently, or you must specify the PARALLEL hint for each insert operation.

To disable direct-path INSERT, specify the NOAPPEND hint in each INSERT statement. Doing so overrides parallel DML mode.

Notes:

- Direct-path INSERT supports only the subquery syntax of the INSERT statement, not the VALUES clause. For more information on the subquery syntax of INSERT statements, see *Oracle Database SQL Reference.*
- There are some additional restrictions for using direct-path INSERT. These are listed in the *Oracle Database SQL Reference.*

See Also: *Oracle Database Performance Tuning Guide* for more information on using hints

How Direct-Path INSERT Works

You can use direct-path INSERT on both partitioned and nonpartitioned tables.

Serial Direct-Path INSERT into Partitioned or Nonpartitioned Tables

The single process inserts data beyond the current high water mark of the table segment or of each partition segment. (The **high-water mark** is the level at which blocks have never been formatted to receive data.) When a COMMIT runs, the high-water mark is updated to the new value, making the data visible to users.

Parallel Direct-Path INSERT into Partitioned Tables

This situation is analogous to serial direct-path INSERT. Each parallel execution server is assigned one or more partitions, with no more than one process working on a single partition. Each parallel execution server inserts data beyond the current high-water mark of its assigned partition segment(s). When a COMMIT runs, the high-water mark of each partition segment is updated to its new value, making the data visible to users.

Parallel Direct-Path INSERT into Nonpartitioned Tables

Each parallel execution server allocates a new temporary segment and inserts data into that temporary segment. When a COMMIT runs, the parallel execution coordinator merges the new temporary segments into the primary table segment, where it is visible to users.

Specifying the Logging Mode for Direct-Path INSERT

Direct-path INSERT lets you choose whether to log redo and undo information during the insert operation.

- You can specify logging mode for a table, partition, index, or LOB storage at create time (in a CREATE statement) or subsequently (in an ALTER statement).
- If you do not specify either LOGGING or NOLOGGING at these times:
  - The logging attribute of a partition defaults to the logging attribute of its table.
  - The logging attribute of a table or index defaults to the logging attribute of the tablespace in which it resides.
  - The logging attribute of LOB storage defaults to LOGGING if you specify CACHE for LOB storage. If you do not specify CACHE, then the logging attributes defaults to that of the tablespace in which the LOB values resides.
- You set the logging attribute of a tablespace in a CREATE TABLESPACE or ALTER TABLESPACE statements.

Note: If the database or tablespace is in FORCE LOGGING mode, then direct path INSERT always logs, regardless of the logging setting.

Direct-Path INSERT with Logging

In this mode, Oracle Database performs full redo logging for instance and media recovery. If the database is in ARCHIVELOG mode, then you can archive redo logs to tape. If the database is in NOARCHIVELOG mode, then you can recover instance crashes but not disk failures.

Direct-Path INSERT without Logging

In this mode, Oracle Database inserts data without redo or undo logging. (Some minimal logging is done to mark new extents invalid, and data dictionary changes are always logged.) This mode improves performance. However, if you subsequently must perform media recovery, the extent invalidation records mark a range of blocks as logically corrupt, because no redo data was logged for them. Therefore, it is important that you back up the data after such an insert operation.

Additional Considerations for Direct-Path INSERT

The following are some additional considerations when using direct-path INSERT.

Index Maintenance with Direct-Path INSERT

Oracle Database performs index maintenance at the end of direct-path INSERT operations on tables (partitioned or nonpartitioned) that have indexes. This index maintenance is performed by the parallel execution servers for parallel direct-path INSERT or by the single process for serial direct-path INSERT. You can avoid the performance impact of index maintenance by dropping the index before the INSERT operation and then rebuilding it afterward.

Space Considerations with Direct-Path INSERT

Direct-path INSERT requires more space than conventional-path INSERT, because direct-path INSERT does not use existing space in the free lists of the segment.

All serial direct-path INSERT operations, as well as parallel direct-path INSERT into partitioned tables, insert data above the high-water mark of the affected segment. This requires some additional space.

Parallel direct-path INSERT into nonpartitioned tables requires even more space, because it creates a temporary segment for each degree of parallelism. If the nonpartitioned table is not in a locally managed tablespace in automatic segment-space management mode, you can modify the values of the NEXT and PCTINCREASE storage parameter and MINIMUM EXTENT tablespace parameter to provide sufficient (but not excess) storage for the temporary segments. Choose values for these parameters so that:

- The size of each extent is not too small (no less than 1 MB). This setting affects the total number of extents in the object.
- The size of each extent is not so large that the parallel INSERT results in wasted space on segments that are larger than necessary.

After the direct-path INSERT operation is complete, you can reset these parameters to settings more appropriate for serial operations.

Locking Considerations with Direct-Path INSERT

During direct-path INSERT, the database obtains exclusive locks on the table (or on all partitions of a partitioned table). As a result, users cannot perform any concurrent insert, update, or delete operations on the table, and concurrent index creation and build operations are not permitted. Concurrent queries, however, are supported, but the query will return only the information before the insert operation.

Automatically Collecting Statistics on Tables

The PL/SQL package DBMS_STATS lets you generate and manage statistics for cost-based optimization. You can use this package to gather, modify, view, export, import, and delete statistics. You can also use this package to identify or name statistics that have been gathered.

Formerly, you enabled DBMS_STATS to automatically gather statistics for a table by specifying the MONITORING keyword in the CREATE (or ALTER) TABLE statement. Starting with Oracle Database 10*g*, the MONITORING and NOMONITORING keywords have been deprecated and statistics are collected automatically. If you do specify these keywords, they are ignored.

Monitoring tracks the approximate number of INSERT, UPDATE, and DELETE operations for the table since the last time statistics were gathered. Information about how many rows are affected is maintained in the SGA, until periodically (about every three hours) SMON incorporates the data into the data dictionary. This data dictionary information is made visible through the DBA_TAB_

MODIFICATIONS, ALL_TAB_MODIFICATIONS, or USER_TAB_MODIFICATIONS views. The database uses these views to identify tables with stale statistics.

To disable monitoring of a table, set the STATISTICS_LEVEL initialization parameter to BASIC. Its default is TYPICAL, which enables automatic statistics collection. Automatic statistics collection and the DBMS_STATS package enable the optimizer to generate accurate execution plans.

See Also:

- *Oracle Database Reference* for detailed information on the STATISTICS_LEVEL initialization parameter
- *Oracle Database Performance Tuning Guide* for information on managing optimizer statistics
- *PL/SQL Packages and Types Reference* for information about using the DBMS_STATS package
- "Automatic Statistics Collection Job" on page 23-2 for information on using the Scheduler to collect statistics automatically

Altering Tables

You alter a table using the ALTER TABLE statement. To alter a table, the table must be contained in your schema, or you must have either the ALTER object privilege for the table or the ALTER ANY TABLE system privilege.

Many of the usages of the ALTER TABLE statement are presented in the following sections:

- Reasons for Using the ALTER TABLE Statement
- Altering Physical Attributes of a Table
- Moving a Table to a New Segment or Tablespace
- Manually Allocating Storage for a Table
- Modifying an Existing Column Definition
- Adding Table Columns
- Renaming Table Columns
- Dropping Table Columns

Caution: Before altering a table, familiarize yourself with the consequences of doing so. The *Oracle Database SQL Reference* lists many of these consequences in the descriptions of the ALTER TABLE clauses.

If a view, materialized view, trigger, domain index, function-based index, check constraint, function, procedure of package depends on a base table, the alteration of the base table or its columns can affect the dependent object. See "Managing Object Dependencies" on page 20-22 for information about how the database manages dependencies.

Reasons for Using the ALTER TABLE Statement

You can use the ALTER TABLE statement to perform any of the following actions that affect a table:

- Modify physical characteristics (PCTFREE, PCTUSED, INITRANS, or storage parameters)
- Move the table to a new segment or tablespace
- Explicitly allocate an extent or deallocate unused space
- Add, drop, or rename columns, or modify an existing column definition (datatype, length, default value, and NOT NULL integrity constraint)
- Modify the logging attributes of the table
- Modify the CACHE / NOCACHE attributes
- Add, modify or drop integrity constraints associated with the table
- Enable or disable integrity constraints or triggers associated with the table
- Modify the degree of parallelism for the table
- Rename a table
- Add or modify index-organized table characteristics
- Alter the characteristics of an external table
- Add or modify LOB columns
- Add or modify object type, nested table, or varray columns

Many of these operations are discussed in succeeding sections.

Altering Physical Attributes of a Table

When altering the data block space usage parameters (PCTFREE and PCTUSED) of a table, note that new settings apply to all data blocks used by the table, including blocks already allocated and subsequently allocated for the table. However, the blocks already allocated for the table are not immediately reorganized when space usage parameters are altered, but as necessary after the change. The data block storage parameters are described in "Managing Space in Data Blocks" on page 13-1.

When altering the transaction entry setting INITRANS of a table, note that a new setting for INITRANS applies only to data blocks subsequently allocated for the table. To better understand this transaction entry setting parameter, see "Specifying the INITRANS Parameter" on page 13-7.

The storage parameters INITIAL and MINEXTENTS cannot be altered. All new settings for the other storage parameters (for example, NEXT, PCTINCREASE) affect only extents subsequently allocated for the table. The size of the next extent allocated is determined by the current values of NEXT and PCTINCREASE, and is not based on previous values of these parameters. Storage parameters are discussed in "Managing Storage Parameters" on page 13-10.

Moving a Table to a New Segment or Tablespace

The ALTER TABLE ... MOVE statement enables you to relocate data of a nonpartitioned table or of a partition of a partitioned table into a new segment, and optionally into a different tablespace for which you have quota. This statement also lets you modify any of the storage attributes of the table or partition, including those which cannot be modified using ALTER TABLE. You can also use the ALTER TABLE ... MOVE statement with the COMPRESS keyword to store the new segment using table compression.

The following statement moves the hr.admin_emp table to a new segment, specifying new storage parameters:

```
ALTER TABLE hr.admin_emp MOVE
      STORAGE ( INITIAL 20K
                NEXT 40K
                MINEXTENTS 2
                MAXEXTENTS 20
                PCTINCREASE 0 );
```

Moving a table changes the rowids of the rows in the table. This causes indexes on the table to be marked UNUSABLE, and DML accessing the table using these indexes will receive an ORA-01502 error. The indexes on the table must be dropped or

rebuilt. Likewise, any statistics for the table become invalid and new statistics should be collected after moving the table.

If the table includes LOB column(s), this statement can be used to move the table along with LOB data and LOB index segments (associated with this table) which the user explicitly specifies. If not specified, the default is to not move the LOB data and LOB index segments.

Manually Allocating Storage for a Table

Oracle Database dynamically allocates additional extents for the data segment of a table, as required. However, perhaps you want to allocate an additional extent for a table explicitly. For example, in an Oracle Real Application Clusters environment, an extent of a table can be allocated explicitly for a specific instance.

A new extent can be allocated for a table using the ALTER TABLE ... ALLOCATE EXTENT clause.

You can also explicitly deallocate unused space using the DEALLOCATE UNUSED clause of ALTER TABLE. This is described in "Reclaiming Unused Space" on page 13-28.

> **See Also:** *Oracle Real Application Clusters Administrator's Guide* for information about using the ALLOCATE EXTENT clause in an Oracle Real Application Clusters environment

Modifying an Existing Column Definition

Use the ALTER TABLE ... MODIFY statement to modify an existing column definition. You can modify column datatype, default value, or column constraint.

You can increase the length of an existing column, or decrease it, if all existing data satisfies the new length. You can change a column from byte semantics to CHAR semantics or vice versa. You must set the initialization parameter BLANK_TRIMMING=TRUE to decrease the length of a nonempty CHAR column.

If you are modifying a table to increase the length of a column of datatype CHAR, realize that this can be a time consuming operation and can require substantial additional storage, especially if the table contains many rows. This is because the CHAR value in each row must be blank-padded to satisfy the new column length.

> **See Also:** *Oracle Database SQL Reference* for additional information about modifying table columns and additional restrictions

Adding Table Columns

To add a column to an existing table, use the ALTER TABLE ... ADD statement.

The following statement alters the hr.admin_emp table to add a new column named bonus:

```
ALTER TABLE hr.admin_emp
      ADD (bonus NUMBER (7,2));
```

If a new column is added to a table, the column is initially NULL unless you specify the DEFAULT clause. When you specify a default value, the database updates each row in the new column with the values specified. Specifying a DEFAULT value is not supported for tables using table compression.

You can add a column with a NOT NULL constraint to a table only if the table does not contain any rows, or you specify a default value.

> **See Also:** *Oracle Database SQL Reference* for additional information about adding table columns and additional restrictions

Renaming Table Columns

Oracle Database lets you rename existing columns in a table. Use the RENAME COLUMN clause of the ALTER TABLE statement to rename a column. The new name must not conflict with the name of any existing column in the table. No other clauses are allowed in conjunction with the RENAME COLUMN clause.

The following statement renames the comm column of the hr.admin_emp table.

```
ALTER TABLE hr.admin_emp
      RENAME COLUMN comm TO commission;
```

As noted earlier, altering a table column can invalidate dependent objects. However, when you rename a column, the database updates associated data dictionary tables to ensure that function-based indexes and check constraints remain valid.

Oracle Database also lets you rename column constraints. This is discussed in "Renaming Constraints" on page 20-17.

> **Note:** The RENAME TO clause of ALTER TABLE appears similar in syntax to the RENAME COLUMN clause, but is used for renaming the table itself.

Dropping Table Columns

You can drop columns that are no longer needed from a table, including an index-organized table. This provides a convenient means to free space in a database, and avoids your having to export/import data then re-create indexes and constraints.

You cannot drop all columns from a table, nor can you drop columns from a table owned by SYS. Any attempt to do so results in an error.

> **See Also:** *Oracle Database SQL Reference* for information about additional restrictions and options for dropping columns from a table

Removing Columns from Tables

When you issue an ALTER TABLE ... DROP COLUMN statement, the column descriptor and the data associated with the target column are removed from each row in the table. You can drop multiple columns with one statement. The ALTER TABLE ... DROP COLUMN statement is not supported for tables using table compression.

The following statements are examples of dropping columns from the hr.admin_emp table. The first statement drops only the sal column:

```
ALTER TABLE hr.admin_emp DROP COLUMN sal;
```

The next statement drops both the bonus and comm columns:

```
ALTER TABLE hr.admin_emp DROP (bonus, commission);
```

Marking Columns Unused

If you are concerned about the length of time it could take to drop column data from all of the rows in a large table, you can use the ALTER TABLE ... SET UNUSED statement. This statement marks one or more columns as unused, but does not actually remove the target column data or restore the disk space occupied by these columns. However, a column that is marked as unused is not displayed in queries or data dictionary views, and its name is removed so that a new column can reuse that name. All constraints, indexes, and statistics defined on the column are also removed.

To mark the hiredate and mgr columns as unused, execute the following statement:

```
ALTER TABLE hr.admin_emp SET UNUSED (hiredate, mgr);
```

You can later remove columns that are marked as unused by issuing an ALTER TABLE ... DROP UNUSED COLUMNS statement. Unused columns are also removed from the target table whenever an explicit drop of any particular column or columns of the table is issued.

The data dictionary views USER_UNUSED_COL_TABS, ALL_UNUSED_COL_TABS, or DBA_UNUSED_COL_TABS can be used to list all tables containing unused columns. The COUNT field shows the number of unused columns in the table.

```
SELECT * FROM DBA_UNUSED_COL_TABS;

OWNER                          TABLE_NAME                     COUNT
------------------------------ ------------------------------ -----
HR                             ADMIN_EMP                          2
```

Removing Unused Columns

The ALTER TABLE ... DROP UNUSED COLUMNS statement is the only action allowed on unused columns. It physically removes unused columns from the table and reclaims disk space.

In the ALTER TABLE statement that follows, the optional clause CHECKPOINT is specified. This clause causes a checkpoint to be applied after processing the specified number of rows, in this case 250. Checkpointing cuts down on the amount of undo logs accumulated during the drop column operation to avoid a potential exhaustion of undo space.

```
ALTER TABLE hr.admin_emp DROP UNUSED COLUMNS CHECKPOINT 250;
```

Redefining Tables Online

In highly available systems, it is occasionally necessary to redefine large "hot" tables to improve the performance of queries or DML performed against these tables. The database provide a mechanism to redefine tables online. This mechanism provides a significant increase in availability compared to traditional methods of redefining tables that require tables to be taken offline.

When a table is redefined online, it is accessible to DML during much of the redefinition process. The table is locked in the exclusive mode only during a very small window which is independent of the size of the table and the complexity of the redefinition.

This section contains the following topics:

Features of Online Table Redefinition

Online table redefinition enables you to:

- Modify the storage parameters of the table
- Move the table to a different tablespace in the same schema
- Add support for parallel queries
- Add or drop partitioning support
- Re-create the table to reduce fragmentation
- Change the organization of a normal table (heap organized) to an index-organized table and vice versa
- Add or drop a column

The DBMS_REDEFINITION Package

The mechanism for performing online redefinition is the PL/SQL package `DBMS_REDEFINITION`. Execute privileges on this package is granted to `EXECUTE_CATALOG_ROLE`. In addition to having execute privileges on this package, you must be granted the following privileges:

- `CREATE ANY TABLE`
- `ALTER ANY TABLE`
- `DROP ANY TABLE`
- `LOCK ANY TABLE`
- `SELECT ANY TABLE`
- `CREATE ANY TRIGGER`

- CREATE ANY INDEX

See Also: *PL/SQL Packages and Types Reference* for a description of the DBMS_REDEFINITION package

Steps for Online Redefinition of Tables

In order to perform an online redefinition of a table the user must perform the following steps.

1. Choose one of the following two methods of redefinition:
   - The first method of redefinition is to use the primary keys or pseudo-primary keys to perform the redefinition. Pseudo-primary keys are unique keys with all component columns having NOT NULL constraints. For this method, the versions of the tables before and after redefinition should have the same primary key columns. This is the preferred and default method of redefinition.
   - The second method of redefinition is to use rowids. For this method, the table to be redefined should not be an index organized table. Also, in this method of redefinition, a hidden column named M_ROW$$ is added to the post-redefined version of the table and it is recommended that this column be marked as unused or dropped after the redefinition is completed.
2. Verify that the table can be online redefined by invoking the DBMS_REDEFINITION.CAN_REDEF_TABLE() procedure and use the OPTIONS_FLAG parameter to specify the method of redefinition to be used. If the table is not a candidate for online redefinition, then this procedure raises an error indicating why the table cannot be online redefined.
3. Create an empty interim table (in the same schema as the table to be redefined) with all of the desired attributes. If columns are to be dropped, do not include them in the definition of the interim table. If a column is to be added, then add the column definition to the interim table.

   It is possible to perform table redefinition in parallel. If you specify a degree of parallelism on both of the tables and you ensure that parallel execution is enabled for the session, the database will use parallel execution whenever possible to perform the redefinition. You can use the PARALLEL clause of the ALTER SESSION statement to enable parallel execution.
4. Start the redefinition process by calling DBMS_REDEFINITION.START_REDEF_TABLE(), providing the following:
   - The table to be redefined

- The interim table name
- The column mapping
- The method of redefinition
- Optionally, the columns to be used in ordering rows
- Optionally, specify the ORDER BY columns

If the column mapping information is not supplied, then it is assumed that all the columns (with their names unchanged) are to be included in the interim table. If the column mapping is supplied, then only those columns specified explicitly in the column mapping are considered. If the method of redefinition is not specified, then the default method of redefinition using primary keys is assumed.

You can optionally specify the ORDERBY_COLS parameter to specify how rows should be ordered during the initial instantiation of the interim table.

5. You have two methods for creating (cloning) dependent objects such as triggers, indexes, grants, and constraints on the interim table. Method 1 is the most automatic and preferred method, but there may be times that you would choose to use method 2.

   - Method 1: Automatically Creating Dependent Objects

     Use the COPY_TABLE_DEPENDENTS procedure to automatically create dependent objects such as triggers, indexes, grants, and constraints on the interim table. This procedure also registers the dependent objects. Registering the dependent objects enables the identities of these objects and their cloned counterparts to be automatically swapped later as part of the redefinition completion process. The result is that when the redefinition is completed, the names of the dependent objects will be the same as the names of the original dependent objects.

     You can discover if errors occurred while copying dependent objects by checking the NUM_ERRORS output variable. If the IGNORE_ERRORS parameter is set to TRUE, the COPY_TABLE_DEPENDENTS procedure continues cloning dependent objects even if an error is encounter when creating an object. The errors can later be viewed by querying the DBA_REDIFINITION_ERRORS view. Reasons for errors include a lack of system resources or a change in the logical structure of the table.

     If IGNORE_ERRORS is set to FALSE, the COPY_TABLE_DEPENDENTS procedure stops cloning objects as soon as any error is encountered.

After you correct any errors you can attempt again to clone the failing object or objects by reexecuting the COPY_TABLE_DEPENDENTS procedure. Optionally you can create the objects manually and then register them as explained in method 2.

The COPY_TABLE_DEPENDENTS procedure can be used multiple times as necessary. If an object has already been successfully cloned, it will ignore the operation.

- Method 2: Manually Creating Dependent Objects

  You can manually create dependent objects on the interim table.

  Note: In previous releases you were *required* to manually create the triggers, indexes, grants, and constraints on the interim table, and there may still be situations where to want to or must do so. In such cases, any referential constraints involving the interim table (that is, the interim table is either a parent or a child table of the referential constraint) must be created disabled. Until the redefinition process is either completed or aborted, any trigger defined on the interim table will not execute.

  Use the REGISTER_DEPENDENT_OBJECT procedure after you create dependent objects manually. You can also use the COPY_TABLE_DEPENDENTS procedure to do the registration. Note that the COPY_TABLE_DEPENDENTS procedure does not clone objects that are registered manually.

  You would also use the REGISTER_DEPENDENT_OBJECT procedure if the COPY_TABLE_DEPENDENTS procedure failed to copy a dependent object and manual intervention is required.

  You can query the DBA_REDEFINITION_OBJECTS view to determine which dependent objects are registered. This view shows dependent objects that were registered explicitly with the REGISTER_DEPENDENT_OBJECT procedure or implicitly with the COPY_TABLE_DEPENDENTS procedure. Only current information is shown in the view.

  The UNREGISTER_DEPENDENT_OBJECT procedure can be used to unregister a dependent object on the table being redefined and on the interim table.

6. Execute the DBMS_REDEFINITION.FINISH_REDEF_TABLE procedure to complete the redefinition of the table. During this procedure, the original table

is locked in the exclusive mode for a very short time, independent of the amount of data in the original table. However, FINISH_REDEF_TABLE will wait for all pending DML that was initiated before it was invoked to commit before completing the redefinition.

As a result of this procedure, the following occur:

a. The original table is redefined such that it has all the attributes, indexes, constraints, grants and triggers of the interim table

b. The referential constraints involving the interim table now involve the post redefined table and are enabled.

c. Dependent objects that were registered, either explicitly using REGISTER_DEPENDENT_OBJECT or implicitly using COPY_TABLE_DEPENDENTS, are renamed automatically.

Note: If no registration is done or no automatic cloning is done, then you must manually rename the dependent objects.

---

7. If the redefinition was done using rowids, the post-redefined table will have a hidden column (M_ROW$$) and it is recommended that the user set this hidden column to unused as follows:

```
ALTER TABLE table_name SET UNUSED (M_ROW$$)
```

The following is the end result of the redefinition process:

- The original table is redefined with the attributes and features of the interim table.
- The triggers, grants, indexes and constraints defined on the interim table after START_REDEF_TABLE( ) and before FINISH_REDEF_TABLE( ) are now defined on the post-redefined table. Any referential constraints involving the interim table before the redefinition process was finished now involve the post-redefinition table and are enabled.
- Any indexes, triggers, grants and constraints defined on the original table (prior to redefinition) are transferred to the interim table and are dropped when the user drops the interim table. Any referential constraints involving the original table before the redefinition now involve the interim table and are disabled.
- Any PL/SQL procedures and cursors defined on the original table (prior to redefinition) are invalidated. They are automatically revalidated (this

revalidation can fail if the shape of the table was changed as a result of the redefinition process) whenever they are used next.

Intermediate Synchronization

After the redefinition process has been started by calling START_REDEF_TABLE( ) and before FINISH_REDEF_TABLE( ) has been called, it is possible that a large number of DML statements have been executed on the original table. If you know this is the case, it is recommended that you periodically synchronize the interim table with the original table. This is done by calling the DBMS_REDEFINITION.SYNC_INTERIM_TABLE( ) procedure. Calling this procedure reduces the time taken by FINISH_REDEF_TABLE( ) to complete the redefinition process.

The small amount of time that the original table is locked during FINISH_REDEF_TABLE( ) is independent of whether SYNC_INTERIM_TABLE( ) has been called.

Terminate and Clean Up After Errors

In the event that an error is raised during the redefinition process, or if you choose to terminate the redefinition process, call DBMS_REDEFINITION.ABORT_REDEF_TABLE( ). This procedure drops temporary logs and tables associated with the redefinition process. After this procedure is called, you can drop the interim table and its associated objects.

Example of Online Table Redefinition

This example illustrates online redefinition of the previously created table hr.admin_emp, which at this point only contains columns: empno, ename, job, deptno. The table is redefined as follows:

- New columns mgr, hiredate, sal, and bonus (these existed in the original table but were dropped in previous examples) are added.
- The new column bonus is initialized to 0
- The column deptno has its value increased by 10.
- The redefined table is partitioned by range on empno.

The steps in this redefinition are illustrated below.

1. Verify that the table is a candidate for online redefinition. In this case you specify that the redefinition is to be done using primary keys or pseudo-primary keys.

```
BEGIN
DBMS_REDEFINITION.CAN_REDEF_TABLE('hr','admin_emp',
       dbms_redefinition.cons_use_pk);
END;
/
```

2. Create an interim table `hr.int_admin_emp`.

```
CREATE TABLE hr.int_admin_emp
       (empno      NUMBER(5) PRIMARY KEY,
         ename      VARCHAR2(15) NOT NULL,
         job        VARCHAR2(10),
         mgr        NUMBER(5),
         hiredate   DATE DEFAULT (sysdate),
         sal        NUMBER(7,2),
         deptno     NUMBER(3) NOT NULL,
         bonus      NUMBER (7,2) DEFAULT(1000))
     PARTITION BY RANGE(empno)
       (PARTITION emp1000 VALUES LESS THAN (1000) TABLESPACE admin_tbs,
        PARTITION emp2000 VALUES LESS THAN (2000) TABLESPACE admin_tbs2);
```

3. Start the redefinition process.

```
BEGIN
DBMS_REDEFINITION.START_REDEF_TABLE('hr', 'admin_emp','int_admin_emp',
       'empno empno, ename ename, job job, deptno+10 deptno, 0 bonus',
        dbms_redefinition.cons_use_pk);
END;
/
```

4. Automatically create any triggers, indexes and constraints on `hr.int_admin_emp`.

```
BEGIN
DBMS_REDEFINITION.COPY_TABLE_DEPENDENTS('hr', 'admin_emp','int_admin_emp',
       TRUE, TRUE, TRUE, FALSE);
END;
```

5. Optionally, synchronize the interim table `hr.int_admin_emp`.

```
BEGIN
DBMS_REDEFINITION.SYNC_INTERIM_TABLE('hr', 'admin_emp', 'int_admin_emp');
END;
/
```

6. Complete the redefinition.

```
BEGIN
DBMS_REDEFINITION.FINISH_REDEF_TABLE('hr', 'admin_emp', 'int_admin_emp');
END;
/
```

The table `hr.admin_emp` is locked in the exclusive mode only for a small window toward the end of this step. After this call the table `hr.admin_emp` is redefined such that it has all the attributes of the `hr.int_admin_emp` table.

7. Drop the interim table.

Restrictions for Online Redefinition of Tables

The following restrictions apply to the online redefinition of tables:

- If the table is to be redefined using primary key or pseudo-primary keys (unique keys or constraints with all component columns having *not null* constraints), then the table to be redefined must have the same primary key or pseudo-primary key columns. If the table is to be redefined using rowids, then the table must not be an index-organized table.
- Tables that are replicated in an n-way master configuration can be redefined, but horizontal subsetting (subset of rows in the table), vertical subsetting (subset of columns in the table), and column transformations are not allowed.
- The overflow table of an index-organized table cannot be online redefined.
- Tables with user-defined types (objects, `REF`s, collections, typed tables) cannot be online redefined.
- Tables with `BFILE` columns cannot be online redefined.
- Tables with `LONG` columns can be online redefined, but those columns must be converted to `CLOB`s. Tables with `LONG RAW` columns must be converted to `BLOB`s. Tables with `LOB` columns are acceptable.
- The table to be redefined cannot be part of a cluster.
- Tables in the `SYS` and `SYSTEM` schema cannot be online redefined.
- Temporary tables cannot be redefined.
- A subset of rows in the table cannot be redefined.
- Only simple deterministic expressions, sequences, and `SYSDATE` can be used when mapping the columns in the interim table to those of the original table. For example, subqueries are not allowed.

- If new columns (which are not instantiated with existing data for the original table) are being added as part of the redefinition, then they must not be declared `NOT NULL` until the redefinition is complete.
- There cannot be any referential constraints between the table being redefined and the interim table.
- Table redefinition cannot be done `NOLOGGING`.
- Tables with materialized view logs defined on them cannot be online redefined.

Auditing Table Changes Using Flashback Transaction Query

Note: You must be using automatic undo management to use the Flashback Transaction Query feature. It is based on undo information stored in an undo tablespace.

To understand how to configure your database for the Flashback Transaction Query feature, see "Monitoring the Undo Tablespace" on page 10-13.

You may discover that somehow data in a table has been inappropriately changed. To research this change, you can use multiple flashback queries to view row data at specific points in time. More efficiently, you can use the Flashback Version Query feature to view all changes to a row over a period of time. That feature lets you append a `VERSIONS` clause to a `SELECT` statement that specifies an SCN or timestamp range between which you want to view changes to row values. The query also can return associated metadata, such as the transaction responsible for the change.

Further, once you identify an erroneous transaction, you can then use the Flashback Transaction Query feature to identify other changes that were done by the transaction, and to request the undo SQL to reverse those changes. But using the undo SQL is only one means of recovering your data. You also have the option of using the Flashback Table feature, described in "Recovering Tables Using the Flashback Table Feature" on page 14-34, to restore the table to a state before the changes were made.

See Also: *Oracle Database Application Developer's Guide - Fundamentals* for directions for using Oracle Flashback Query and Flashback Version Query.

Recovering Tables Using the Flashback Table Feature

> **Note:** You must be using automatic undo management to use the Flashback Table feature. It is based on undo information stored in an undo tablespace.
>
> To understand how to configure your undo tablespace for the Flashback Table feature, see "Monitoring the Undo Tablespace" on page 10-13.

The FLASHBACK TABLE statement enables users to recover a table to a previous point in time. It provides a fast, online solution for recovering a table that has been accidentally modified or deleted by a user or application. In many cases, this Flashback Table feature alleviates the need for you, as the administrator, to perform more complicated point in time recovery operations.

The functionality of the Flashback Table feature can be summarized as follows:

- Restores all data in a specified table to a previous point in time described by a timestamp or SCN.
- Performs the restore operation online.
- Automatically maintains all of the table attributes, such as indexes, triggers, and constraints that are necessary for an application to function with the flashed-back table.
- Maintains any remote state in a distributed environment. For example, all of the table modifications required by replication if a replicated table is flashed back.
- Maintains data integrity as specified by constraints. Tables are flashed back provided none of the table constraints are violated. This includes any referential integrity constraints specified between a table included in the FLASHBACK TABLE statement and another table that is not included in the FLASHBACK TABLE statement.
- Even after a flashback operation, the data in the original table is not lost. You can later revert to the original state.

> **See Also:** *Oracle Database Backup and Recovery Advanced User's Guide* for more information about the FLASHBACK TABLE statement.

Dropping Tables

To drop a table that you no longer need, use the DROP TABLE statement. The table must be contained in your schema or you must have the DROP ANY TABLE system privilege.

Caution: Before dropping a table, familiarize yourself with the consequences of doing so:

- Dropping a table removes the table definition from the data dictionary. All rows of the table are no longer accessible.
- All indexes and triggers associated with a table are dropped.
- All views and PL/SQL program units dependent on a dropped table remain, yet become invalid (not usable). See "Managing Object Dependencies" on page 20-22 for information about how the database manages dependencies.
- All synonyms for a dropped table remain, but return an error when used.
- All extents allocated for a table that is dropped are returned to the free space of the tablespace and can be used by any other object requiring new extents or new objects. All rows corresponding to a clustered table are deleted from the blocks of the cluster. Clustered tables are the subject of Chapter 17, "Managing Clusters".

The following statement drops the hr.int_admin_emp table:

```
DROP TABLE hr.int_admin_emp;
```

If the table to be dropped contains any primary or unique keys referenced by foreign keys of other tables and you intend to drop the FOREIGN KEY constraints of the child tables, then include the CASCADE clause in the DROP TABLE statement, as shown below:

```
DROP TABLE hr.admin_emp CASCADE CONSTRAINTS;
```

When you drop a table, normally the database does not immediately release the space associated with the table. Rather, the database renames the table and places it in a recycle bin, where it can later be recovered with the FLASHBACK TABLE statement if you find that you dropped the table in error. If you should want to

immediately release the space associated with the table at the time you issue the DROP TABLE statement, include the PURGE clause as shown in the following statement:

```
DROP TABLE hr.admin_emp PURGE;
```

Perhaps instead of dropping a table, you want to truncate it. The TRUNCATE statement provides a fast, efficient method for deleting all rows from a table, but it does not affect any structures associated with the table being truncated (column definitions, constraints, triggers, and so forth) or authorizations. The TRUNCATE statement is discussed in "Truncating Tables and Clusters" on page 20-7.

Using Flashback Drop and Managing the Recycle Bin

When you drop a table, the database does not immediately remove the space associated with the table. The database renames the table and places it and any associated objects in a recycle bin, where, in case the table was dropped in error, it can be recovered at a later time. This feature is called Flashback Drop, and the FLASHBACK TABLE statement is used to restore the table. Before discussing the use of the FLASHBACK TABLE statement for this purpose, it is important to understand how the recycle bin works, and how you manage its contents.

This section contains the following topics:

- What Is the Recycle Bin?
- Renaming Objects in the Recycle Bin
- Viewing and Querying Objects in the Recycle Bin
- Purging Objects in the Recycle Bin
- Restoring Tables from the Recycle Bin

What Is the Recycle Bin?

The recycle bin is actually a data dictionary table containing information about dropped objects. Dropped tables and any associated objects such as indexes, constraints, nested tables, and the likes are not removed and still occupy space. They continue to count against user space quotas, until specifically purged from the recycle bin or the unlikely situation where they must be purged by the database because of tablespace space constraints.

Each user can be thought of as having his own recycle bin, since unless a user has the SYSDBA privilege, the only objects that the user has access to in the recycle bin

are those that the user owns. A user can view his objects in the recycle bin using the following statement:

```
SELECT * FROM RECYCLEBIN;
```

When you drop a tablespace including its contents, the objects in the tablespace are not placed in the recycle bin and the database purges any entries in the recycle bin for objects located in the tablespace. The database also purges any recycle bin entries for objects in a tablespace when you drop the tablespace, not including contents, and the tablespace is otherwise empty. Likewise:

- When you drop a user, any objects belonging to the user are not placed in the recycle bin and any objects in the recycle bin are purged.
- When you drop a cluster, its member tables are not placed in the recycle bin and any former member tables in the recycle bin are purged.
- When you drop a type, any dependent objects such as subtypes are not placed in the recycle bin and any former dependent objects in the recycle bin are purged.

Renaming Objects in the Recycle Bin

When a dropped table is moved to the recycle bin, the table and its associated objects are given system-generated names. This is necessary to avoid name conflicts that may arise if multiple tables have the same name. This could occur under the following circumstances:

- A user drops a table, re-creates it with the same name, then drops it again.
- Two users have tables with the same name, and both users drop their tables.

The renaming convention is as follows:

```
BIN$unique_id$version
```

where:

- *unique_id* is a 26-character globally unique identifier for this object, which makes the recycle bin name unique across all databases
- *version* is a version number assigned by the database

Viewing and Querying Objects in the Recycle Bin

Oracle Database provides two views for obtaining information about objects in the recycle bin:

View	Description
USER_RECYCLEBIN	This view can be used by users to see their own dropped objects in the recycle bin. It has a synonym RECYCLEBIN, for ease of use.
DBA_RECYCLEBIN	This view gives administrators visibility to all dropped objects in the recycle bin

One use for these views is to identify the name that the database has assigned to a dropped object, as shown in the following example:

```
SELECT object_name, original_name FROM dba_recyclebin
   WHERE owner = 'HR';

OBJECT_NAME                    ORIGINAL_NAME
------------------------------ --------------------------------
BIN$yrMKlZaLMhfgNAgAIMenRA==$0 EMPLOYEES
```

You can also view the contents of the recycle bin using the SQL*Plus command SHOW RECYCLEBIN.

```
SQL> show recyclebin

ORIGINAL NAME    RECYCLEBIN NAME                OBJECT TYPE  DROP TIME
---------------- ------------------------------ ------------ -------------------
EMPLOYEES        BIN$yrMKlZaVMhfgNAgAIMenRA==$0 TABLE        2003-10-27:14:00:19
```

You can query objects that are in the recycle bin, just as you can query other objects. However, you must specify the name of the object as it is identified in the recycle bin. For example:

```
SELECT * FROM "BIN$yrMKlZaVMhfgNAgAIMenRA==$0";
```

Purging Objects in the Recycle Bin

If you decide that you are never going to restore an item from the recycle bin, you can use the PURGE statement to remove the items and their associated objects from the recycle bin and release their storage space. You need the same privileges as if you were dropping the item.

When you use the PURGE statement to purge a table, you can use the name that the table is known by in the recycle bin or the original name of the table. The recycle bin name can be obtained from either the DBA_ or USER_RECYCLEBIN view as shown in "Viewing and Querying Objects in the Recycle Bin" on page 14-38. The following hypothetical example purges the table hr.int_admin_emp, which was renamed to BIN$jsleilx392mk2=293$0 when it was placed in the recycle bin:

```
PURGE TABLE BIN$jsleilx392mk2=293$0;
```

You can achieve the same result with the following statement:

```
PURGE TABLE int_admin_emp;
```

You can use the PURGE statement to purge all the objects in the recycle bin that are from a specified tablespace or only the tablespace objects belonging to a specified user, as shown in the following examples:

```
PURGE TABLESPACE example;
PURGE TABLESPACE example USER oe;
```

Users can purge the recycle bin or their own objects, and release space for objects, by using the following statement:

```
PURGE RECYCLEBIN;
```

If you have the SYSDBA privilege, then you can purge the entire recycle bin by specifying DBA_RECYCLEBIN, instead of RECYCLEBIN in the previous statement.

You can also use the PURGE statement to purge an index from the recycle bin or to purge from the recycle bin all objects in a specified tablespace.

> **See Also:** *Oracle Database SQL Reference* for more information on the PURGE statement

Restoring Tables from the Recycle Bin

Use the FLASHBACK TABLE ... TO BEFORE DROP statement to recover objects from the recycle bin. You can specify either the name of the table in the recycle bin or the original table name. An optional RENAME TO clause lets you rename the table as you recover it. The recycle bin name can be obtained from either the DBA_ or USER_RECYCLEBIN view as shown in "Viewing and Querying Objects in the Recycle Bin" on page 14-38. To use the FLASHBACK TABLE ... TO BEFORE DROP statement, you need the same privileges you need to drop the table.

The following example restores `int_admin_emp` table and assigns to it a new name:

```
FLASHBACK TABLE int_admin_emp TO BEFORE DROP
   RENAME TO int2_admin_emp;
```

The system-generated recycle bin name is very useful if you have dropped a table multiple times. For example, suppose you have three versions of the `int2_admin_emp` table in the recycle bin and you want to recover the second version. You can do this by issuing two FLASHBACK TABLE statements, or you can query the recycle bin and then flashback to the appropriate system-generated name, as shown in the following example:

```
SELECT object_name, original_name FROM recyclebin;

OBJECT_NAME                    ORIGINAL_NAME
------------------------------ --------------------------------
BIN$yrMKlZaLMhfgNAgAIMenRA==$0 INT2_ADMIN_EMP
BIN$yrMKlZaVMhfgNAgAIMenRA==$0 INT2_ADMIN_EMP
BIN$yrMKlZaQMhfgNAgAIMenRA==$0 INT2_ADMIN_EMP

FLASHBACK TABLE BIN$yrMKlZaVMhfgNAgAIMenRA==$0 TO BEFORE DROP
   RENAME TO int2_admin_emp;
```

Managing Index-Organized Tables

This section describes aspects of managing index-organized tables, and contains the following topics:

- What Are Index-Organized Tables?
- Creating Index-Organized Tables
- Maintaining Index-Organized Tables
- Creating Secondary Indexes on Index-Organized Tables
- Analyzing Index-Organized Tables
- Using the ORDER BY Clause with Index-Organized Tables
- Converting Index-Organized Tables to Regular Tables

What Are Index-Organized Tables?

An **index-organized table** has a storage organization that is a variant of a primary B-tree. Unlike an ordinary (heap-organized) table whose data is stored as an unordered collection (heap), data for an index-organized table is stored in a B-tree index structure in a primary key sorted manner. Each leaf block in the index structure stores both the key and nonkey columns.

The structure of an index-organized table provides the following benefits:

- Fast random access on the primary key because an index-only scan is sufficient. And, because there is no separate table storage area, changes to the table data (such as adding new rows, updating rows, or deleting rows) result only in updating the index structure.
- Fast range access on the primary key because the rows are clustered in primary key order.
- Lower storage requirements because duplication of primary keys is avoided. They are not stored both in the index and underlying table, as is true with heap-organized tables.

Index-organized tables have full table functionality. They support features such as constraints, triggers, LOB and object columns, partitioning, parallel operations, online reorganization, and replication. And, they offer these additional features:

- Key compression
- Overflow storage area and specific column placement
- Secondary indexes, including bitmap indexes.

Index-organized tables are ideal for OLTP applications, which require fast primary key access and high availability. Queries and DML on an orders table used in electronic order processing are predominantly primary-key based and heavy volume causes fragmentation resulting in a frequent need to reorganize. Because an index-organized table can be reorganized online and without invalidating its secondary indexes, the window of unavailability is greatly reduced or eliminated.

Index-organized tables are suitable for modeling application-specific index structures. For example, content-based information retrieval applications containing text, image and audio data require inverted indexes that can be effectively modeled using index-organized tables. A fundamental component of an internet search engine is an inverted index that can be modeled using index-organized tables.

These are but a few of the applications for index-organized tables.

See Also:

- *Oracle Database Concepts* for a more thorough description of index-organized tables
- Chapter 16, "Managing Partitioned Tables and Indexes" contains information about partitioning index-organized tables

Creating Index-Organized Tables

You use the `CREATE TABLE` statement to create index-organized tables, but you must provide additional information:

- An `ORGANIZATION INDEX` qualifier, which indicates that this is an index-organized table
- A primary key, specified through a column constraint clause (for a single column primary key) or a table constraint clause (for a multiple-column primary key).

Optionally, you can specify the following:

- An `OVERFLOW` clause, which preserves dense clustering of the B-tree index by storing the row column values exceeding a specified threshold in a separate overflow data segment.
- A `PCTTHRESHOLD` value, which defines the percentage of space reserved in the index block for an index-organized table. Any portion of the row that exceeds the specified threshold is stored in the overflow segment. In other words, the row is broken at a column boundary into two pieces, a head piece and tail piece. The head piece fits in the specified threshold and is stored along with the key in the index leaf block. The tail piece is stored in the overflow area as one or more row pieces. Thus, the index entry contains the key value, the nonkey column values that fit the specified threshold, and a pointer to the rest of the row.
- An `INCLUDING` clause, which can be used to specify nonkey columns that are to be stored in the overflow data segment.

Creating an Index-Organized Table

The following statement creates an index-organized table:

```
CREATE TABLE admin_docindex(
        token char(20),
        doc_id NUMBER,
        token_frequency NUMBER,
        token_offsets VARCHAR2(512),
```

```
        CONSTRAINT pk_admin_docindex PRIMARY KEY (token, doc_id))
    ORGANIZATION INDEX
    TABLESPACE admin_tbs
    PCTTHRESHOLD 20
    OVERFLOW TABLESPACE admin_tbs2;
```

Specifying `ORGANIZATION INDEX` causes the creation of an index-organized table, `admin_docindex`, where the key columns and nonkey columns reside in an index defined on columns that designate the primary key or keys for the table. In this case, the primary keys are `token` and `doc_id`. An overflow segment is specified and is discussed in "Using the Overflow Clause" on page 14-44.

Creating Index-Organized Tables that Contain Object Types

Index-organized tables can store object types. The following example creates object type `admin_typ`, then creates an index-organized table containing a column of object type `admin_typ`:

```
CREATE OR REPLACE TYPE admin_typ AS OBJECT
    (col1 NUMBER, col2 VARCHAR2(6));
CREATE TABLE admin_iot (c1 NUMBER primary key, c2 admin_typ)
    ORGANIZATION INDEX;
```

You can also create an index-organized table of object types. For example:

```
CREATE TABLE admin_iot2 OF admin_typ (col1 PRIMARY KEY)
    ORGANIZATION INDEX;
```

Another example, that follows, shows that index-organized tables store nested tables efficiently. For a nested table column, the database internally creates a storage table to hold all the nested table rows.

```
CREATE TYPE project_t AS OBJECT(pno NUMBER, pname VARCHAR2(80));
/
CREATE TYPE project_set AS TABLE OF project_t;
/
CREATE TABLE proj_tab (eno NUMBER, projects PROJECT_SET)
    NESTED TABLE projects STORE AS emp_project_tab
                ((PRIMARY KEY(nested_table_id, pno))
    ORGANIZATION INDEX)
    RETURN AS LOCATOR;
```

The rows belonging to a single nested table instance are identified by a `nested_table_id` column. If an ordinary table is used to store nested table columns, the nested table rows typically get de-clustered. But when you use an index-organized

table, the nested table rows can be clustered based on the `nested_table_id` column.

See Also:

- *Oracle Database SQL Reference* for details of the syntax used for creating index-organized tables
- "Creating Partitioned Index-Organized Tables" on page 16-25 for information about creating partitioned index-organized tables
- *Oracle Database Application Developer's Guide - Object-Relational Features* for information about object types

Using the Overflow Clause

The overflow clause specified in the statement shown in "Creating an Index-Organized Table" on page 14-42 indicates that any nonkey columns of rows exceeding 20% of the block size are placed in a data segment stored in the `admin_tbs2` tablespace. The key columns should fit the specified threshold.

If an update of a nonkey column causes the row to decrease in size, the database identifies the row piece (head or tail) to which the update is applicable and rewrites that piece.

If an update of a nonkey column causes the row to increase in size, the database identifies the piece (head or tail) to which the update is applicable and rewrites that row piece. If the target of the update turns out to be the head piece, note that this piece can again be broken into two to keep the row size below the specified threshold.

The nonkey columns that fit in the index leaf block are stored as a row head-piece that contains a `rowid` field linking it to the next row piece stored in the overflow data segment. The only columns that are stored in the overflow area are those that do not fit.

Choosing and Monitoring a Threshold Value

You should choose a threshold value that can accommodate your key columns, as well as the first few nonkey columns (if they are frequently accessed).

After choosing a threshold value, you can monitor tables to verify that the value you specified is appropriate. You can use the `ANALYZE TABLE ... LIST CHAINED ROWS` statement to determine the number and identity of rows exceeding the threshold value.

See Also:

- "Listing Chained Rows of Tables and Clusters" on page 20-5 for more information about chained rows
- *Oracle Database SQL Reference* for syntax of the ANALYZE statement

Using the INCLUDING Clause

In addition to specifying PCTTHRESHOLD, you can use the INCLUDING clause to control which nonkey columns are stored with the key columns. The database accommodates all nonkey columns up to the column specified in the INCLUDING clause in the index leaf block, provided it does not exceed the specified threshold. All nonkey columns beyond the column specified in the INCLUDING clause are stored in the overflow area.

Note: Oracle Database moves all primary key columns of an indexed-organized table to the beginning of the table (in their key order), in order to provide efficient primary key based access. As an example:

```
CREATE TABLE admin_iot4(a INT, b INT, c INT, d INT,
                primary key(c,b))
    ORGANIZATION INDEX;
```

The stored column order is: c b a d (instead of: a b c d). The last primary key column is b, based on the stored column order. The INCLUDING column can be the last primary key column (b in this example), or any nonkey column (that is, any column after b in the stored column order).

The following CREATE TABLE statement is similar to the one shown earlier in "Creating an Index-Organized Table" on page 14-42 but is modified to create an index-organized table where the token_offsets column value is always stored in the overflow area:

```
CREATE TABLE admin_docindex2(
        token CHAR(20),
        doc_id NUMBER,
        token_frequency NUMBER,
        token_offsets VARCHAR2(512),
        CONSTRAINT pk_admin_docindex2 PRIMARY KEY (token, doc_id))
    ORGANIZATION INDEX
```

```
    TABLESPACE admin_tbs
    PCTTHRESHOLD 20
    INCLUDING token_frequency
    OVERFLOW TABLESPACE admin_tbs2;
```

Here, only nonkey columns prior to `token_offsets` (in this case a single column only) are stored with the key column values in the index leaf block.

Parallelizing Index-Organized Table Creation

The `CREATE TABLE ... AS SELECT` statement enables you to create an index-organized table and load data from an existing table into it. By including the `PARALLEL` clause, the load can be done in parallel.

The following statement creates an index-organized table in parallel by selecting rows from the conventional table `hr.jobs`:

```
CREATE TABLE admin_iot3(i PRIMARY KEY, j, k, l)
     ORGANIZATION INDEX
     PARALLEL
     AS SELECT * FROM hr.jobs;
```

This statement provides an alternative to parallel bulk-load using SQL*Loader.

Using Key Compression

Creating an index-organized table using key compression enables you to eliminate repeated occurrences of key column prefix values.

Key compression breaks an index key into a prefix and a suffix entry. Compression is achieved by sharing the prefix entries among all the suffix entries in an index block. This sharing can lead to huge savings in space, allowing you to store more keys in each index block while improving performance.

You can enable key compression using the `COMPRESS` clause while:

- Creating an index-organized table
- Moving an index-organized table

You can also specify the prefix length (as the number of key columns), which identifies how the key columns are broken into a prefix and suffix entry.

```
CREATE TABLE admin_iot5(i INT, j INT, k INT, l INT, PRIMARY KEY (i, j, k))
    ORGANIZATION INDEX COMPRESS;
```

The preceding statement is equivalent to the following statement:

```
CREATE TABLE admin_iot6(i INT, j INT, k INT, l INT, PRIMARY KEY(i, j, k))
    ORGANIZATION INDEX COMPRESS 2;
```

For the list of values (1,2,3), (1,2,4), (1,2,7), (1,3,5), (1,3,4), (1,4,4) the repeated occurrences of (1,2), (1,3) are compressed away.

You can also override the default prefix length used for compression as follows:

```
CREATE TABLE admin_iot7(i INT, j INT, k INT, l INT, PRIMARY KEY (i, j, k))
    ORGANIZATION INDEX COMPRESS 1;
```

For the list of values (1,2,3), (1,2,4), (1,2,7), (1,3,5), (1,3,4), (1,4,4), the repeated occurrences of 1 are compressed away.

You can disable compression as follows:

```
ALTER TABLE admin_iot5 MOVE NOCOMPRESS;
```

One application of key compression is in a time-series application that uses a set of time-stamped rows belonging to a single item, such as a stock price. Index-organized tables are attractive for such applications because of the ability to cluster rows based on the primary key. By defining an index-organized table with primary key (stock symbol, time stamp), you can store and manipulate time-series data efficiently. You can achieve more storage savings by compressing repeated occurrences of the item identifier (for example, the stock symbol) in a time series by using an index-organized table with key compression.

> **See Also:** *Oracle Database Concepts* for more information about key compression

Maintaining Index-Organized Tables

Index-organized tables differ from ordinary tables only in physical organization. Logically, they are manipulated in the same manner as ordinary tables. You can specify an index-organized table just as you would specify a regular table in INSERT, SELECT, DELETE, and UPDATE statements.

Altering Index-Organized Tables

All of the alter options available for ordinary tables are available for index-organized tables. This includes ADD, MODIFY, and DROP COLUMNS and CONSTRAINTS. However, the primary key constraint for an index-organized table cannot be dropped, deferred, or disabled

You can use the ALTER TABLE statement to modify physical and storage attributes for both primary key index and overflow data segments. All the attributes specified prior to the OVERFLOW keyword are applicable to the primary key index segment. All attributes specified after the OVERFLOW key word are applicable to the overflow data segment. For example, you can set the INITRANS of the primary key index segment to 4 and the overflow of the data segment INITRANS to 6 as follows:

```
ALTER TABLE admin_docindex INITRANS 4 OVERFLOW INITRANS 6;
```

You can also alter PCTTHRESHOLD and INCLUDING column values. A new setting is used to break the row into head and overflow tail pieces during subsequent operations. For example, the PCTHRESHOLD and INCLUDING column values can be altered for the admin_docindex table as follows:

```
ALTER TABLE admin_docindex PCTTHRESHOLD 15 INCLUDING doc_id;
```

By setting the INCLUDING column to doc_id, all the columns that follow token_frequency and token_offsets, are stored in the overflow data segment.

For index-organized tables created without an overflow data segment, you can add an overflow data segment by using the ADD OVERFLOW clause. For example, you can add an overflow segment to table admin_iot3 as follows:

```
ALTER TABLE admin_iot3 ADD OVERFLOW TABLESPACE admin_tbs2;
```

Moving (Rebuilding) Index-Organized Tables

Because index-organized tables are primarily stored in a B-tree index, you can encounter fragmentation as a consequence of incremental updates. However, you can use the ALTER TABLE ... MOVE statement to rebuild the index and reduce this fragmentation.

The following statement rebuilds the index-organized table admin_docindex:

```
ALTER TABLE admin_docindex MOVE;
```

You can rebuild index-organized tables online using the ONLINE keyword. The overflow data segment, if present, is rebuilt when the OVERFLOW keyword is specified. For example, to rebuild the admin_docindex table but not the overflow data segment, perform a move online as follows:

```
ALTER TABLE admin_docindex MOVE ONLINE;
```

To rebuild the admin_docindex table along with its overflow data segment perform the move operation as shown in the following statement. This statement

also illustrates moving both the table and overflow data segment to new tablespaces.

```
ALTER TABLE admin_docindex MOVE TABLESPACE admin_tbs2
    OVERFLOW TABLESPACE admin_tbs3;
```

In this last statement, an index organized table with a LOB column (CLOB) is created. Later, the table is moved with the LOB index and data segment being rebuilt and moved to a new tablespace.

```
CREATE TABLE admin_iot_lob
   (c1 number (6) primary key,
    admin_lob CLOB)
   ORGANIZATION INDEX
   LOB (admin_lob) STORE AS (TABLESPACE admin_tbs2);
.
.
.
ALTER TABLE admin_iot_lob MOVE LOB (admin_lob) STORE AS (TABLESPACE admin_tbs3);
```

> **See Also:** *Oracle Database Application Developer's Guide - Large Objects* contains information about LOBs in index-organized tables

Creating Secondary Indexes on Index-Organized Tables

You can create secondary indexes on an index organized tables to provide multiple access paths. Secondary indexes on index-organized tables differ from indexes on ordinary tables in two ways:

- They store logical rowids instead of physical rowids. This is necessary because the inherent movability of rows in a B-tree index results in the rows having no permanent physical addresses. If the physical location of a row changes, its logical rowid remains valid. One effect of this is that a table maintenance operation, such as ALTER TABLE ... MOVE, does not make the secondary index unusable.
- The logical rowid also includes a physical guess which identifies the database block address at which the row is likely to be found. If the physical guess is correct, a secondary index scan would incur a single additional I/O once the secondary key is found. The performance would be similar to that of a secondary index-scan on an ordinary table.

Unique and nonunique secondary indexes, function-based secondary indexes, and bitmap indexes are supported as secondary indexes on index-organized tables.

Creating a Secondary Index on an Index-Organized Table

The following statement shows the creation of a secondary index on the docindex index-organized table where doc_id and token are the key columns:

```
CREATE INDEX Doc_id_index on Docindex(Doc_id, Token);
```

This secondary index allows the database to efficiently process a query, such as the following, the involves a predicate on doc_id:

```
SELECT Token FROM Docindex WHERE Doc_id = 1;
```

Maintaining Physical Guesses in Logical Rowids

A logical rowid can include a guess, which identifies the block location of a row at the time the guess is made. Instead of doing a full key search, the database uses the guess to search the block directly. However, as new rows are inserted, guesses can become stale. The indexes are still usable through the primary key-component of the logical rowid, but access to rows is slower.

Collect index statistics with the DBMS_STATS package to monitor the staleness of guesses. The database checks whether the existing guesses are still valid and records the percentage of rows with valid guesses in the data dictionary. This statistic is stored in the PCT_DIRECT_ACCESS column of the DBA_INDEXES view (and related views).

To obtain fresh guesses, you can rebuild the secondary index. Note that rebuilding a secondary index on an index-organized table involves reading the base table, unlike rebuilding an index on an ordinary table. A quicker, more light weight means of fixing the guesses is to use the ALTER INDEX ... UPDATE BLOCK REFERENCES statement. This statement is performed online, while DML is still allowed on the underlying index-organized table.

After you rebuild a secondary index, or otherwise update the block references in the guesses, collect index statistics again.

Bitmap Indexes

Bitmap indexes on index-organized tables are supported, provided the index-organized table is created with a mapping table. This is done by specifying the MAPPING TABLE clause in the CREATE TABLE statement that you use to create the index-organized table, or in an ALTER TABLE statement to add the mapping table later.

> **See Also:** *Oracle Database Concepts* for a description of mapping tables

Analyzing Index-Organized Tables

Just like ordinary tables, index-organized tables are analyzed using the DBMS_STATS package, or the ANALYZE statement.

Collecting Optimizer Statistics for Index-Organized Tables

To collect optimizer statistics, use the DBMS_STATS package.

For example, the following statement gathers statistics for the index-organized countries table in the hr schema:

```
EXECUTE DBMS_STATS.GATHER_TABLE_STATS ('HR','COUNTRIES');
```

The DBMS_STATS package analyzes both the primary key index segment and the overflow data segment, and computes logical as well as physical statistics for the table.

- The logical statistics can be queried using USER_TABLES, ALL_TABLES or DBA_TABLES.
- You can query the physical statistics of the primary key index segment using USER_INDEXES, ALL_INDEXES or DBA_INDEXES (and using the primary key index name). For example, you can obtain the primary key index segment physical statistics for the table admin_docindex as follows:

  ```
  SELECT LAST_ANALYZED, BLEVEL,LEAF_BLOCKS, DISTINCT_KEYS
     FROM DBA_INDEXES WHERE INDEX_NAME= 'PK_ADMIN_DOCINDEX';
  ```

- You can query the physical statistics for the overflow data segment using the USER_TABLES, ALL_TABLES or DBA_TABLES. You can identify the overflow entry by searching for IOT_TYPE = 'IOT_OVERFLOW'. For example, you can obtain overflow data segment physical attributes associated with the admin_docindex table as follows:

  ```
  SELECT LAST_ANALYZED, NUM_ROWS, BLOCKS, EMPTY_BLOCKS
     FROM DBA_TABLES WHERE IOT_TYPE='IOT_OVERFLOW'
           and IOT_NAME= 'ADMIN_DOCINDEX';
  ```

See Also:

- *Oracle Database Performance Tuning Guide* for more information about collecting optimizer statistics
- *PL/SQL Packages and Types Reference* for more information about of the DBMS_STATS package

Validating the Structure of Index-Organized Tables

Use the ANALYZE statement if you want to validate the structure of your index-organized table or to list any chained rows. These operations are discussed in the following sections located elsewhere in this book:

- "Validating Tables, Indexes, Clusters, and Materialized Views" on page 20-4
- "Listing Chained Rows of Tables and Clusters" on page 20-5

> **Note:** There are special considerations when listing chained rows for index-organized tables. These are discussed in the *Oracle Database SQL Reference*.

Using the ORDER BY Clause with Index-Organized Tables

If an ORDER BY clause only references the primary key column or a prefix of it, then the optimizer avoids the sorting overhead, as the rows are returned sorted on the primary key columns.

The following queries avoid sorting overhead because the data is already sorted on the primary key:

```
SELECT * FROM admin_docindex2 ORDER BY token, doc_id;
SELECT * FROM admin_docindex2 ORDER BY token;
```

If, however, you have an ORDER BY clause on a suffix of the primary key column or non-primary-key columns, additional sorting is required (assuming no other secondary indexes are defined).

```
SELECT * FROM admin_docindex2 ORDER BY doc_id;
SELECT * FROM admin_docindex2 ORDER BY token_frequency;
```

Converting Index-Organized Tables to Regular Tables

You can convert index-organized tables to regular tables using the Oracle import or export utilities, or the CREATE TABLE ... AS SELECT statement.

To convert an index-organized table to a regular table:

- Export the index-organized table data using conventional path.
- Create a regular table definition with the same definition.
- Import the index-organized table data, making sure IGNORE=y (ensures that object exists error is ignored).

Note: Before converting an index-organized table to a regular table, be aware that index-organized tables cannot be exported using pre-Oracle8 versions of the Export utility.

See Also: *Oracle Database Utilities* for more details about using the IMPORT and EXPORT utilities

Managing External Tables

Oracle Database allows you read-only access to data in external tables. External tables are defined as tables that do not reside in the database, and can be in any format for which an access driver is provided. By providing the database with metadata describing an external table, the database is able to expose the data in the external table as if it were data residing in a regular database table. The external data can be queried directly and in parallel using SQL.

You can, for example, select, join, or sort external table data. You can also create views and synonyms for external tables. However, no DML operations (UPDATE, INSERT, or DELETE) are possible, and no indexes can be created, on external tables.

External tables also provide a framework to unload the result of an arbitrary SELECT statement into a platform-independent Oracle-proprietary format that can be used by Oracle Data Pump.

Note: The DBMS_STATS package can be used for gathering statistics for external tables. The ANALYZE statement is not supported for gathering statistics for external tables.

For information about using the DBMS_STATS package, see *Oracle Database Performance Tuning Guide*

The means of defining the metadata for external tables is through the CREATE TABLE ... ORGANIZATION EXTERNAL statement. This external table definition can be thought of as a view that allows running any SQL query against external data without requiring that the external data first be loaded into the database. An access driver is the actual mechanism used to read the external data in the table. When you use external tables to unload data, the metadata is automatically created based on the datatypes in the SELECT statement (sometimes referred to as the shape of the query).

Oracle Database provides two access drivers for external tables. The default access driver is ORACLE_LOADER, which allows the reading of data from external files using the Oracle loader technology. The ORACLE_LOADER access driver provides data mapping capabilities which are a subset of the control file syntax of SQL*Loader utility. The second access driver, ORACLE_DATAPUMP, lets you unload data--that is, read data from the database and insert it into an external table, represented by one or more external files--and then reload it into an Oracle Database.

The Oracle Database external tables feature provides a valuable means for performing basic extraction, transformation, and loading (ETL) tasks that are common for data warehousing.

These following sections discuss the DDL statements that are supported for external tables. Only DDL statements discussed are supported, and not all clauses of these statements are supported.

- Creating External Tables
- Altering External Tables
- Dropping External Tables
- System and Object Privileges for External Tables

See Also:

- *Oracle Database Utilities* contains more information about external tables and describes the access drivers and their access parameters
- *Oracle Data Warehousing Guide* for information about using external tables in a data warehousing environment

Creating External Tables

You create external tables using the ORGANIZATION EXTERNAL clause of the CREATE TABLE statement. You are not in fact creating a table; that is, an external table does not have any extents associated with it. Rather, you are creating metadata in the data dictionary that enables you to access external data.

The following example creates an external table and then uploads the data to a database table. Alternatively, you can unload data through the external table framework by specifying the AS *subquery* clause of the CREATE TABLE statement. External table data pump unload can use only the ORACLE_DATAPUMP access driver.

EXAMPLE: Creating an External Table and Loading Data

The file `empxt1.dat` contains the following sample data:

```
360,Jane,Janus,ST_CLERK,121,17-MAY-2001,3000,0,50,jjanus
361,Mark,Jasper,SA_REP,145,17-MAY-2001,8000,.1,80,mjasper
362,Brenda,Starr,AD_ASST,200,17-MAY-2001,5500,0,10,bstarr
363,Alex,Alda,AC_MGR,145,17-MAY-2001,9000,.15,80,aalda
```

The file `empxt2.dat` contains the following sample data:

```
401,Jesse,Cromwell,HR_REP,203,17-MAY-2001,7000,0,40,jcromwel
402,Abby,Applegate,IT_PROG,103,17-MAY-2001,9000,.2,60,aapplega
403,Carol,Cousins,AD_VP,100,17-MAY-2001,27000,.3,90,ccousins
404,John,Richardson,AC_ACCOUNT,205,17-MAY-2001,5000,0,110,jrichard
```

The following hypothetical SQL statements create an external table in the `hr` schema named `admin_ext_employees` and load its data into the `hr.employees` table.

```
CONNECT  /  AS SYSDBA;
-- Set up directories and grant access to hr
CREATE OR REPLACE DIRECTORY admin_dat_dir
    AS '/flatfiles/data';
CREATE OR REPLACE DIRECTORY admin_log_dir
    AS '/flatfiles/log';
CREATE OR REPLACE DIRECTORY admin_bad_dir
    AS '/flatfiles/bad';
GRANT READ ON DIRECTORY admin_dat_dir TO hr;
GRANT WRITE ON DIRECTORY admin_log_dir TO hr;
GRANT WRITE ON DIRECTORY admin_bad_dir TO hr;
-- hr connects
CONNECT hr/hr
-- create the external table
CREATE TABLE admin_ext_employees
                  (employee_id       NUMBER(4),
                   first_name        VARCHAR2(20),
                   last_name         VARCHAR2(25),
                   job_id            VARCHAR2(10),
                   manager_id        NUMBER(4),
                   hire_date         DATE,
                   salary            NUMBER(8,2),
                   commission_pct    NUMBER(2,2),
                   department_id     NUMBER(4),
                   email             VARCHAR2(25)
                  )
```

```
     ORGANIZATION EXTERNAL
     (
       TYPE ORACLE_LOADER
       DEFAULT DIRECTORY admin_dat_dir
       ACCESS PARAMETERS
       (
         records delimited by newline
         badfile admin_bad_dir:'empxt%a_%p.bad'
         logfile admin_log_dir:'empxt%a_%p.log'
         fields terminated by ','
         missing field values are null
         ( employee_id, first_name, last_name, job_id, manager_id,
           hire_date char date_format date mask "dd-mon-yyyy",
           salary, commission_pct, department_id, email
         )
       )
       LOCATION ('empxt1.dat', 'empxt2.dat')
     )
     PARALLEL
     REJECT LIMIT UNLIMITED;
-- enable parallel for loading (good if lots of data to load)
ALTER SESSION ENABLE PARALLEL DML;
-- load the data in hr employees table
INSERT INTO employees (employee_id, first_name, last_name, job_id, manager_id,
                       hire_date, salary, commission_pct, department_id, email)
            SELECT * FROM admin_ext_employees;
```

The following paragraphs contain descriptive information about this example.

The first few statements in this example create the directory objects for the operating system directories that contain the data sources, and for the bad record and log files specified in the access parameters. You must also grant `READ` or `WRITE` directory object privileges, as appropriate.

Note: When creating a directory object or BFILEs, ensure that the following conditions are met:

- The operating system file must not be a symbolic or hard link.
- The operating system directory path named in the Oracle Database directory object must be an existing OS directory path.
- The operating system directory path named in the directory object should not contain any symbolic links in its components.

The TYPE specification indicates the access driver of the external table. The access driver is the API that interprets the external data for 5the database. Oracle Database provides two access drivers: ORACLE_LOADER and ORACLE_DATAPUMP. If you omit the TYPE specification, ORACLE_LOADER is the default access driver. You must specify the ORACLE_DATAPUMP access driver if you specify the AS *subquery* clause to unload data from one Oracle Database and reload it into the same or a different Oracle Database.

The access parameters, specified in the ACCESS PARAMETERS clause, are opaque to the database. These access parameters are defined by the access driver, and are provided to the access driver by the database when the external table is accessed. See *Oracle Database Utilities* for a description of the ORACLE_LOADER access parameters.

The PARALLEL clause enables parallel query on the data sources. The granule of parallelism is by default a data source, but parallel access within a data source is implemented whenever possible. For example, if PARALLEL=3 were specified, then more than one parallel execution server could be working on a data source. But, parallel access within a data source is provided by the access driver only if all of the following conditions are met:

- The media allows random positioning within a data source
- It is possible to find a record boundary from a random position
- The data files are large enough to make it worthwhile to break up into multiple chunks

Note: Specifying a PARALLEL clause is of value *only* when dealing with large amounts of data. Otherwise, it is not advisable to specify a PARALLEL clause, and doing so can be detrimental.

The REJECT LIMIT clause specifies that there is no limit on the number of errors that can occur during a query of the external data. For parallel access, this limit applies to each parallel execution server independently. For example, if REJECT LIMIT is specified, each parallel query process is allowed 10 rejections. Hence, the only precisely enforced values for REJECT LIMIT on parallel query are 0 and UNLIMITED.

In this example, the INSERT INTO TABLE statement generates a dataflow from the external data source to the Oracle Database SQL engine where data is processed. As data is parsed by the access driver from the external table sources and provided to the external table interface, the external data is converted from its external representation to its Oracle Database internal datatype.

See Also: *Oracle Database SQL Reference* provides details of the syntax of the CREATE TABLE statement for creating external tables and specifies restrictions on the use of clauses

Altering External Tables

You can use any of the ALTER TABLE clauses shown in Table 14–1to change the characteristics of an external table. No other clauses are permitted.

Table 14–1 Altering an External Table

ALTER TABLE Clause	Description	Example
REJECT LIMIT	Changes the reject limit	`ALTER TABLE admin_ext_employees REJECT LIMIT 100;`

Table 14–1 (Cont.) Altering an External Table

ALTER TABLE Clause	Description	Example

PROJECT COLUMN	Determines how the access driver validates rows in subsequent queries:
	▪ PROJECT COLUMN REFERENCED: the access driver processes only the columns in the select list of the query. This setting may not provide a consistent set of rows when querying a different column list from the same external table. This is the default.
	▪ PROJECT COLUMN ALL: the access driver processes all of the columns defined on the external table. This setting always provides a consistent set of rows when querying an external table.
		`ALTER TABLE admin_ext_employees PROJECT COLUMN REFERNCED;`
		
		`ALTER TABLE admin_ext_employees PROJECT COLUMN ALL;`

DEFAULT DIRECTORY	Changes the default directory specification	`ALTER TABLE admin_ext_employees DEFAULT DIRECTORY admin_dat2_dir;`
ACCESS PARAMETERS	Allows access parameters to be changed without dropping and re-creating the external table metadata	`ALTER TABLE admin_ext_employees ACCESS PARAMETERS (FIELDS TERMINATED BY ';');`
LOCATION	Allows data sources to be changed without dropping and re-creating the external table metadata	`ALTER TABLE admin_ext_employees LOCATION ('empxt3.txt', 'empxt4.txt');`
PARALLEL	No difference from regular tables. Allows degree of parallelism to be changed.	No new syntax
ADD COLUMN	No difference from regular tables. Allows a column to be added to an external table.	No new syntax
MODIFY COLUMN	No difference from regular tables. Allows an external table column to be modified.	No new syntax
DROP COLUMN	No difference from regular tables. Allows an external table column to be dropped.	No new syntax
RENAME TO	No difference from regular tables. Allows external table to be renamed.	No new syntax

Dropping External Tables

For an external table, the DROP TABLE statement removes only the table metadata in the database. It has no affect on the actual data, which resides outside of the database.

System and Object Privileges for External Tables

System and object privileges for external tables are a subset of those for regular table. Only the following system privileges are applicable to external tables:

- CREATE ANY TABLE
- ALTER ANY TABLE
- DROP ANY TABLE
- SELECT ANY TABLE

Only the following object privileges are applicable to external tables:

- ALTER
- SELECT

However, object privileges associated with a directory are:

- READ
- WRITE

For external tables, READ privileges are required on directory objects that contain data sources, while WRITE privileges are required for directory objects containing bad, log, or discard files.

Viewing Information About Tables

The following views allow you to access information about tables.

View	Description
DBA_TABLES ALL_TABLES USER_TABLES	DBA view describes all relational tables in the database. ALL view describes all tables accessible to the user. USER view is restricted to tables owned by the user. Some columns in these views contain statistics that are generated by the DBMS_STATS package or ANALYZE statement.

View	Description
DBA_TAB_COLUMNS ALL_TAB_COLUMNS USER_TAB_COLUMNS	These views describe the columns of tables, views, and clusters in the database. Some columns in these views contain statistics that are generated by the DBMS_STATS package or ANALYZE statement.
DBA_ALL_TABLES ALL_ALL_TABLES USER_ALL_TABLES	These views describe all relational and object tables in the database. Object tables are not specifically discussed in this book.
DBA_TAB_COMMENTS ALL_TAB_COMMENTS USER_TAB_COMMENTS	These views display comments for tables and views. Comments are entered using the COMMENT statement.
DBA_COL_COMMENTS ALL_COL_COMMENTS USER_COL_COMMENTS	These views display comments for table and view columns. Comments are entered using the COMMENT statement.
DBA_EXTERNAL_TABLES ALL_EXTERNAL_TABLES USER_EXTERNAL_TABLES	These views list the specific attributes of external tables in the database.
DBA_EXTERNAL_LOCATIONS ALL_EXTERNAL_LOCATIONS USER_EXTERNAL_ LOCATIONS	These views list the data sources for external tables.
DBA_TAB_HISTOGRAMS ALL_TAB_HISTOGRAMS USER_TAB_HISTOGRAMS	These views describe histograms on tables and views.
DBA_TAB_COL_STATISTICS ALL_TAB_COL_STATISTICS USER_TAB_COL_ STATISTICS	These views provide column statistics and histogram information extracted from the related TAB_COLUMNS views.
DBA_TAB_MODIFICATIONS ALL_TAB_MODIFICATIONS USER_TAB_MODIFICATIONS	These views describe tables that have been modified since the last time table statistics were gathered on them. They are not populated immediately, but after a time lapse (usually 3 hours).

View	Description
DBA_UNUSED_COL_TABS ALL_UNUSED_COL_TABS USER_UNUSED_COL_TABS	These views list tables with unused columns, as marked by the ALTER TABLE ... SET UNUSED statement.
DBA_PARTIAL_DROP_TABS ALL_PARTIAL_DROP_TABS USER_PARTIAL_DROP_TABS	These views list tables that have partially completed DROP COLUMN operations. These operations could be incomplete because the operation was interrupted by the user or a system failure.

Example: Displaying Column Information

Column information, such as name, datatype, length, precision, scale, and default data values can be listed using one of the views ending with the _COLUMNS suffix. For example, the following query lists all of the default column values for the emp and dept tables:

```
SELECT TABLE_NAME, COLUMN_NAME, DATA_TYPE, DATA_LENGTH, LAST_ANALYZED
    FROM DBA_TAB_COLUMNS
    WHERE OWNER = 'HR'
    ORDER BY TABLE_NAME;
```

The following is the output from the query:

```
TABLE_NAME           COLUMN_NAME          DATA_TYPE   DATA_LENGTH LAST_ANALYZED
-------------------- -------------------- ---------- ------------ -------------
COUNTRIES            COUNTRY_ID           CHAR                2 05-FEB-03
COUNTRIES            COUNTRY_NAME         VARCHAR2           40 05-FEB-03
COUNTRIES            REGION_ID            NUMBER             22 05-FEB-03
DEPARTMENTS          DEPARTMENT_ID        NUMBER             22 05-FEB-03
DEPARTMENTS          DEPARTMENT_NAME      VARCHAR2           30 05-FEB-03
DEPARTMENTS          MANAGER_ID           NUMBER             22 05-FEB-03
DEPARTMENTS          LOCATION_ID          NUMBER             22 05-FEB-03
EMPLOYEES            EMPLOYEE_ID          NUMBER             22 05-FEB-03
EMPLOYEES            FIRST_NAME           VARCHAR2           20 05-FEB-03
EMPLOYEES            LAST_NAME            VARCHAR2           25 05-FEB-03
EMPLOYEES            EMAIL                VARCHAR2           25 05-FEB-03
.
.
.
LOCATIONS            COUNTRY_ID           CHAR                2 05-FEB-03
REGIONS              REGION_ID            NUMBER             22 05-FEB-03
REGIONS              REGION_NAME          VARCHAR2           25 05-FEB-03
```

```
51 rows selected.
```

See Also:

- *Oracle Database Reference* for complete descriptions of these views
- *Oracle Database Application Developer's Guide - Object-Relational Features* for information about object tables
- *Oracle Database Performance Tuning Guide* for information about histograms and generating statistics for tables
- "Analyzing Tables, Indexes, and Clusters" on page 20-2

15

Managing Indexes

This chapter discusses the management of indexes, and contains the following topics:

About Indexes

Indexes are optional structures associated with tables and clusters that allow SQL statements to execute more quickly against a table. Just as the index in this manual helps you locate information faster than if there were no index, an Oracle Database index provides a faster access path to table data. You can use indexes without rewriting any queries. Your results are the same, but you see them more quickly.

Oracle Database provides several indexing schemes that provide complementary performance functionality. These are:

- B-tree indexes: the default and the most common
- B-tree cluster indexes: defined specifically for cluster
- Hash cluster indexes: defined specifically for a hash cluster
- Global and local indexes: relate to partitioned tables and indexes

- Reverse key indexes: most useful for Oracle Real Application Clusters applications
- Bitmap indexes: compact; work best for columns with a small set of values
- Function-based indexes: contain the precomputed value of a function / expression
- Domain indexes: specific to an application or cartridge.

Indexes are logically and physically independent of the data in the associated table. Being independent structures, they require storage space. You can create or drop an index without affecting the base tables, database applications, or other indexes. The database automatically maintains indexes when you insert, update, and delete rows of the associated table. If you drop an index, all applications continue to work. However, access to previously indexed data might be slower.

> **See Also:** Chapter 13, "Managing Space for Schema Objects" is recommended reading before attempting tasks described in this chapter.

Guidelines for Managing Indexes

This section discusses guidelines for managing indexes and contains the following topics:

See Also:

- *Oracle Database Concepts* for conceptual information about indexes and indexing, including descriptions of the various indexing schemes offered by Oracle
- *Oracle Database Performance Tuning Guide* and *Oracle Data Warehousing Guide* for information about bitmap indexes
- *Oracle Data Cartridge Developer's Guide* for information about defining domain-specific operators and indexing schemes and integrating them into the Oracle Database server

Create Indexes After Inserting Table Data

Data is often inserted or loaded into a table using either the SQL*Loader or an import utility. It is more efficient to create an index for a table after inserting or loading the data. If you create one or more indexes before loading data, the database then must update every index as each row is inserted.

Creating an index on a table that already has data requires sort space. Some sort space comes from memory allocated for the index creator. The amount for each user is determined by the initialization parameter `SORT_AREA_SIZE`. The database also swaps sort information to and from temporary segments that are only allocated during the index creation in the users temporary tablespace.

Under certain conditions, data can be loaded into a table with SQL*Loader direct-path load and an index can be created as data is loaded.

See Also: *Oracle Database Utilities* for information about using SQL*Loader for direct-path load

Index the Correct Tables and Columns

Use the following guidelines for determining when to create an index:

- Create an index if you frequently want to retrieve less than 15% of the rows in a large table. The percentage varies greatly according to the relative speed of a table scan and how the distribution of the row data in relation to the index key. The faster the table scan, the lower the percentage; the more clustered the row data, the higher the percentage.
- To improve performance on joins of multiple tables, index columns used for joins.

> **Note:** Primary and unique keys automatically have indexes, but you might want to create an index on a foreign key.

- Small tables do not require indexes. If a query is taking too long, then the table might have grown from small to large.

Some columns are strong candidates for indexing. Columns with one or more of the following characteristics are candidates for indexing:

- Values are relatively unique in the column.
- There is a wide range of values (good for regular indexes).
- There is a small range of values (good for bitmap indexes).
- The column contains many nulls, but queries often select all rows having a value. In this case, use the following phrase:

  ```
  WHERE COL_X > -9.99 * power(10,125)
  ```

  Using the preceding phrase is preferable to:

  ```
  WHERE COL_X IS NOT NULL
  ```

  This is because the first uses an index on `COL_X` (assuming that `COL_X` is a numeric column).

Columns with the following characteristics are less suitable for indexing:

- There are many nulls in the column and you do not search on the not null values.

`LONG` and `LONG RAW` columns cannot be indexed.

The size of a single index entry cannot exceed roughly one-half (minus some overhead) of the available space in the data block.

Order Index Columns for Performance

The order of columns in the `CREATE INDEX` statement can affect query performance. In general, specify the most frequently used columns first.

If you create a single index across columns to speed up queries that access, for example, `col1`, `col2`, and `col3`; then queries that access just `col1`, or that access just `col1` and `col2`, are also speeded up. But a query that accessed just `col2`, just `col3`, or just `col2` and `col3` does not use the index.

Limit the Number of Indexes for Each Table

A table can have any number of indexes. However, the more indexes there are, the more overhead is incurred as the table is modified. Specifically, when rows are inserted or deleted, all indexes on the table must be updated as well. Also, when a column is updated, all indexes that contain the column must be updated.

Thus, there is a trade-off between the speed of retrieving data from a table and the speed of updating the table. For example, if a table is primarily read-only, having more indexes can be useful; but if a table is heavily updated, having fewer indexes could be preferable.

Drop Indexes That Are No Longer Required

Consider dropping an index if:

- It does not speed up queries. The table could be very small, or there could be many rows in the table but very few index entries.
- The queries in your applications do not use the index.
- The index must be dropped before being rebuilt.

See Also: "Monitoring Index Usage" on page 15-18

Specify Index Block Space Use

When an index is created for a table, data blocks of the index are filled with the existing values in the table up to PCTFREE. The space reserved by PCTFREE for an index block is only used when a new row is inserted into the table and the corresponding index entry must be placed in the correct index block (that is, between preceding and following index entries).

If no more space is available in the appropriate index block, the indexed value is placed where it belongs (based on the lexical set ordering). Therefore, if you plan on inserting many rows into an indexed table, PCTFREE should be high to accommodate the new index values. If the table is relatively static without many inserts, PCTFREE for an associated index can be low so that fewer blocks are required to hold the index data.

PCTUSED cannot be specified for indexes.

See Also: "Managing Space in Data Blocks" on page 13-1 for information about the PCTFREE parameter

Estimate Index Size and Set Storage Parameters

Estimating the size of an index before creating one can facilitate better disk space planning and management. You can use the combined estimated size of indexes, along with estimates for tables, the undo tablespace, and redo log files, to determine the amount of disk space that is required to hold an intended database. From these estimates, you can make correct hardware purchases and other decisions.

Use the estimated size of an individual index to better manage the disk space that the index uses. When an index is created, you can set appropriate storage parameters and improve I/O performance of applications that use the index. For example, assume that you estimate the maximum size of an index before creating it. If you then set the storage parameters when you create the index, fewer extents are allocated for the table data segment, and all of the index data is stored in a relatively contiguous section of disk space. This decreases the time necessary for disk I/O operations involving this index.

The maximum size of a single index entry is approximately one-half the data block size.

> **See Also:** "Managing Storage Parameters" on page 13-10 for specific information about storage parameters

Specify the Tablespace for Each Index

Indexes can be created in any tablespace. An index can be created in the same or different tablespace as the table it indexes. If you use the same tablespace for a table and its index, it can be more convenient to perform database maintenance (such as tablespace or file backup) or to ensure application availability. All the related data is always online together.

Using different tablespaces (on different disks) for a table and its index produces better performance than storing the table and index in the same tablespace. Disk contention is reduced. But, if you use different tablespaces for a table and its index and one tablespace is offline (containing either data or index), then the statements referencing that table are not guaranteed to work.

Consider Parallelizing Index Creation

You can parallelize index creation, much the same as you can parallelize table creation. Because multiple processes work together to create the index, the database can create the index more quickly than if a single server process created the index sequentially.

When creating an index in parallel, storage parameters are used separately by each query server process. Therefore, an index created with an INITIAL value of 5M and a parallel degree of 12 consumes at least 60M of storage during index creation.

See Also:

- *Oracle Database Concepts* for more information about parallel execution
- *Oracle Data Warehousing Guide* for information about utilizing parallel execution in a data warehousing environment

Consider Creating Indexes with NOLOGGING

You can create an index and generate minimal redo log records by specifying NOLOGGING in the CREATE INDEX statement.

Note: Because indexes created using NOLOGGING are not archived, perform a backup after you create the index.

Creating an index with NOLOGGING has the following benefits:

- Space is saved in the redo log files.
- The time it takes to create the index is decreased.
- Performance improves for parallel creation of large indexes.

In general, the relative performance improvement is greater for larger indexes created without LOGGING than for smaller ones. Creating small indexes without LOGGING has little effect on the time it takes to create an index. However, for larger indexes the performance improvement can be significant, especially when you are also parallelizing the index creation.

Consider Costs and Benefits of Coalescing or Rebuilding Indexes

Improper sizing or increased growth can produce index fragmentation. To eliminate or reduce fragmentation, you can rebuild or coalesce the index. But before you perform either task weigh the costs and benefits of each option and choose the one that works best for your situation. Table 15–1 is a comparison of the costs and benefits associated with rebuilding and coalescing indexes.

Table 15–1 To Rebuild or Coalesce ... That Is the Question

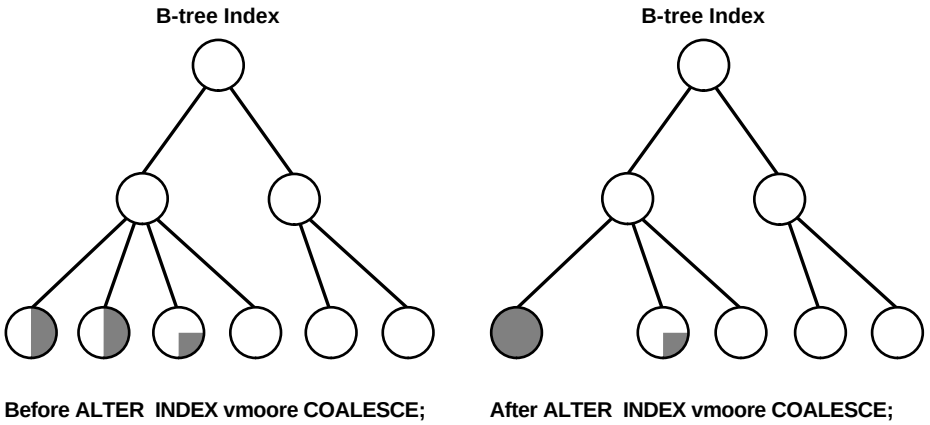
Rebuild Index	Coalesce Index
Quickly moves index to another tablespace	Cannot move index to another tablespace
Higher costs: requires more disk space	Lower costs: does not require more disk space
Creates new tree, shrinks height if applicable	Coalesces leaf blocks within same branch of tree
Enables you to quickly change storage and tablespace parameters without having to drop the original index.	Quickly frees up index leaf blocks for use.

In situations where you have B-tree index leaf blocks that can be freed up for reuse, you can merge those leaf blocks using the following statement:

```
ALTER INDEX vmoore COALESCE;
```

Figure 15–1 illustrates the effect of an `ALTER INDEX COALESCE` on the index `vmoore`. Before performing the operation, the first two leaf blocks are 50% full. This means you have an opportunity to reduce fragmentation and completely fill the first block, while freeing up the second. In this example, assume that `PCTFREE=0`.

Figure 15–1 Coalescing Indexes



Consider Cost Before Disabling or Dropping Constraints

Because unique and primary keys have associated indexes, you should factor in the cost of dropping and creating indexes when considering whether to disable or drop a `UNIQUE` or `PRIMARY KEY` constraint. If the associated index for a `UNIQUE` key or `PRIMARY KEY` constraint is extremely large, you can save time by leaving the constraint enabled rather than dropping and re-creating the large index. You also have the option of explicitly specifying that you want to keep or drop the index when dropping or disabling a `UNIQUE` or `PRIMARY KEY` constraint.

See Also: "Managing Integrity Constraints" on page 20-12

Creating Indexes

This section describes how to create indexes. To create an index in your own schema, *at least one* of the following conditions must be true:

- The table or cluster to be indexed is in your own schema.
- You have `INDEX` privilege on the table to be indexed.
- You have `CREATE ANY INDEX` system privilege.

To create an index in another schema, *all* of the following conditions must be true:

- You have `CREATE ANY INDEX` system privilege.
- The owner of the other schema has a quota for the tablespaces to contain the index or index partitions, or `UNLIMITED TABLESPACE` system privilege.

This section contains the following topics:

Creating an Index Explicitly

You can create indexes explicitly (outside of integrity constraints) using the SQL statement CREATE INDEX. The following statement creates an index named emp_ename for the ename column of the emp table:

```
CREATE INDEX emp_ename ON emp(ename)
      TABLESPACE users
      STORAGE (INITIAL 20K
      NEXT 20k
      PCTINCREASE 75)
      PCTFREE 0;
```

Notice that several storage settings and a tablespace are explicitly specified for the index. If you do not specify storage options (such as INITIAL and NEXT) for an index, the default storage options of the default or specified tablespace are automatically used.

> **See Also:** *Oracle Database SQL Reference* for syntax and restrictions on the use of the CREATE INDEX statement

Creating a Unique Index Explicitly

Indexes can be unique or nonunique. Unique indexes guarantee that no two rows of a table have duplicate values in the key column (or columns). Nonunique indexes do not impose this restriction on the column values.

Use the CREATE UNIQUE INDEX statement to create a unique index. The following example creates a unique index:

```
CREATE UNIQUE INDEX dept_unique_index ON dept (dname)
      TABLESPACE indx;
```

Alternatively, you can define UNIQUE integrity constraints on the desired columns. The database enforces UNIQUE integrity constraints by automatically defining a unique index on the unique key. This is discussed in the following section. However, it is advisable that any index that exists for query performance, including unique indexes, be created explicitly.

> **See Also:** *Oracle Database Performance Tuning Guide* for more information about creating an index for performance

Creating an Index Associated with a Constraint

Oracle Database enforces a UNIQUE key or PRIMARY KEY integrity constraint on a table by creating a unique index on the unique key or primary key. This index is automatically created by the database when the constraint is enabled. No action is required by you when you issue the CREATE TABLE or ALTER TABLE statement to create the index, but you can optionally specify a USING INDEX clause to exercise control over its creation. This includes both when a constraint is defined and enabled, and when a defined but disabled constraint is enabled.

To enable a UNIQUE or PRIMARY KEY constraint, thus creating an associated index, the owner of the table must have a quota for the tablespace intended to contain the index, or the UNLIMITED TABLESPACE system privilege. The index associated with a constraint always takes the name of the constraint, unless you optionally specify otherwise.

Note: An efficient procedure for enabling a constraint that can make use of parallelism is described in"Efficient Use of Integrity Constraints: A Procedure" on page 20-14.

Specifying Storage Options for an Index Associated with a Constraint

You can set the storage options for the indexes associated with UNIQUE and PRIMARY KEY constraints using the USING INDEX clause. The following CREATE TABLE statement enables a PRIMARY KEY constraint and specifies the storage options of the associated index:

```
CREATE TABLE emp (
     empno NUMBER(5) PRIMARY KEY, age INTEGER)
     ENABLE PRIMARY KEY USING INDEX
     TABLESPACE users
     PCTFREE 0;
```

Specifying the Index Associated with a Constraint

If you require more explicit control over the indexes associated with UNIQUE and PRIMARY KEY constraints, the database lets you:

- Specify an existing index that the database is to use to enforce the constraint
- Specify a CREATE INDEX statement that the database is to use to create the index and enforce the constraint

These options are specified using the USING INDEX clause. The following statements present some examples.

Example 1:

```
CREATE TABLE a (
     a1 INT PRIMARY KEY USING INDEX (create index ai on a (a1)));
```

Example 2:

```
CREATE TABLE b(
     b1 INT,
     b2 INT,
     CONSTRAINT bu1 UNIQUE (b1, b2)
                    USING INDEX (create unique index bi on b(b1, b2)),
     CONSTRAINT bu2 UNIQUE (b2, b1) USING INDEX bi);
```

Example 3:

```
CREATE TABLE c(c1 INT, c2 INT);
CREATE INDEX ci ON c (c1, c2);
ALTER TABLE c ADD CONSTRAINT cpk PRIMARY KEY (c1) USING INDEX ci;
```

If a single statement creates an index with one constraint and also uses that index for another constraint, the system will attempt to rearrange the clauses to create the index before reusing it.

See Also: "Managing Integrity Constraints" on page 20-12

Collecting Incidental Statistics when Creating an Index

Oracle Database provides you with the opportunity to collect statistics at very little resource cost during the creation or rebuilding of an index. These statistics are stored in the data dictionary for ongoing use by the optimizer in choosing a plan for the execution of SQL statements. The following statement computes index, table, and column statistics while building index emp_ename on column ename of table emp:

```
CREATE INDEX emp_ename ON emp(ename)
     COMPUTE STATISTICS;
```

See Also:

- *Oracle Database Performance Tuning Guide* for information about collecting statistics and their use by the optimizer
- "Analyzing Tables, Indexes, and Clusters" on page 20-2

Creating a Large Index

When creating an extremely large index, consider allocating a larger temporary tablespace for the index creation using the following procedure:

1. Create a new temporary tablespace using the `CREATE TABLESPACE` or `CREATE TEMPORARY TABLESPACE` statement.
2. Use the `TEMPORARY TABLESPACE` option of the `ALTER USER` statement to make this your new temporary tablespace.
3. Create the index using the `CREATE INDEX` statement.
4. Drop this tablespace using the `DROP TABLESPACE` statement. Then use the `ALTER USER` statement to reset your temporary tablespace to your original temporary tablespace.

Using this procedure can avoid the problem of expanding your usual, and usually shared, temporary tablespace to an unreasonably large size that might affect future performance.

Creating an Index Online

You can create and rebuild indexes online. This enables you to update base tables at the same time you are building or rebuilding indexes on that table. You can perform DML operations while the index build is taking place, but DDL operations are not allowed. Parallel execution is not supported when creating or rebuilding an index online.

The following statements illustrate online index build operations:

```
CREATE INDEX emp_name ON emp (mgr, emp1, emp2, emp3) ONLINE;
```

Note: While you can perform DML operations during an online index build, Oracle recommends that you do not perform major/large DML operations during this procedure. This is because while the DML on the base table is taking place it holds a lock on that resource. The DDL to build the index cannot proceed until the transaction acting on the base table commits or rolls back, thus releasing the lock.

For example, if you want to load rows that total up to 30% of the size of an existing table, you should perform this load before the online index build.

See Also: "Rebuilding an Existing Index" on page 15-18

Creating a Function-Based Index

Function-based indexes facilitate queries that qualify a value returned by a function or expression. The value of the function or expression is precomputed and stored in the index.

To create a function-based index, you must have the `COMPATIBLE` parameter set to 8.1.0.0.0 or higher. In addition to the prerequisites for creating a conventional index, if the index is based on user-defined functions, then those functions must be marked `DETERMINISTIC`. Also, you just have the `EXECUTE` object privilege on any user-defined function(s) used in the function-based index if those functions are owned by another user.

Additionally, to use a function-based index:

- The table must be analyzed after the index is created.
- The query must be guaranteed not to need any `NULL` values from the indexed expression, since `NULL` values are not stored in indexes.

Note: `CREATE INDEX` stores the timestamp of the most recent function used in the function-based index. This timestamp is updated when the index is validated. When performing tablespace point-in-time recovery of a function-based index, if the timestamp on the most recent function used in the index is newer than the timestamp stored in the index, then the index is marked invalid. You must use the `ANALYZE INDEX ... VALIDATE STRUCTURE` statement to validate this index.

To illustrate a function-based index, consider the following statement that defines a function-based index (`area_index`) defined on the function `area(geo)`:

```
CREATE INDEX area_index ON rivers (area(geo));
```

In the following SQL statement, when `area(geo)` is referenced in the `WHERE` clause, the optimizer considers using the index `area_index`.

```
SELECT id, geo, area(geo), desc
     FROM rivers
     WHERE Area(geo) >5000;
```

Table owners should have `EXECUTE` privileges on the functions used in function-based indexes.

Because a function-based index depends upon any function it is using, it can be invalidated when a function changes. If the function is valid, you can use an `ALTER INDEX ... ENABLE` statement to enable a function-based index that has been disabled. The `ALTER INDEX ... DISABLE` statement lets you disable the use of a function-based index. Consider doing this if you are working on the body of the function.

See Also:

- *Oracle Database Concepts* for more information about function-based indexes
- *Oracle Database Application Developer's Guide - Fundamentals* for information about using function-based indexes in applications and examples of their use

Creating a Key-Compressed Index

Creating an index using key compression enables you to eliminate repeated occurrences of key column prefix values.

Key compression breaks an index key into a prefix and a suffix entry. Compression is achieved by sharing the prefix entries among all the suffix entries in an index block. This sharing can lead to huge savings in space, allowing you to store more keys for each index block while improving performance.

Key compression can be useful in the following situations:

- You have a nonunique index where ROWID is appended to make the key unique. If you use key compression here, the duplicate key is stored as a prefix entry on the index block without the ROWID. The remaining rows become suffix entries consisting of only the ROWID.
- You have a unique multicolumn index.

You enable key compression using the COMPRESS clause. The prefix length (as the number of key columns) can also be specified to identify how the key columns are broken into a prefix and suffix entry. For example, the following statement compresses duplicate occurrences of a key in the index leaf block:

```
CREATE INDEX  emp_ename ON emp(ename)
   TABLESPACE users
   COMPRESS 1;
```

The COMPRESS clause can also be specified during rebuild. For example, during rebuild you can disable compression as follows:

```
ALTER INDEX emp_ename REBUILD NOCOMPRESS;
```

> **See Also:** *Oracle Database Concepts* for a more detailed discussion of key compression

Altering Indexes

To alter an index, your schema must contain the index or you must have the ALTER ANY INDEX system privilege. Among the actions allowed by the ALTER INDEX statement are:

- Rebuild or coalesce an existing index
- Deallocate unused space or allocate a new extent
- Specify parallel execution (or not) and alter the degree of parallelism

- Alter storage parameters or physical attributes
- Specify LOGGING or NOLOGGING
- Enable or disable key compression
- Mark the index unusable
- Start or stop the monitoring of index usage

You cannot alter index column structure.

More detailed discussions of some of these operations are contained in the following sections:

- Altering Storage Characteristics of an Index
- Rebuilding an Existing Index
- Monitoring Index Usage

Altering Storage Characteristics of an Index

Alter the storage parameters of any index, including those created by the database to enforce primary and unique key integrity constraints, using the ALTER INDEX statement. For example, the following statement alters the emp_ename index:

```
ALTER INDEX emp_ename
     STORAGE (PCTINCREASE 50);
```

The storage parameters INITIAL and MINEXTENTS cannot be altered. All new settings for the other storage parameters affect only extents subsequently allocated for the index.

For indexes that implement integrity constraints, you can adjust storage parameters by issuing an ALTER TABLE statement that includes the USING INDEX subclause of the ENABLE clause. For example, the following statement changes the storage options of the index created on table emp to enforce the primary key constraint:

```
ALTER TABLE emp
     ENABLE PRIMARY KEY USING INDEX
     PCTFREE 5;
```

> **See Also:** *Oracle Database SQL Reference* for syntax and restrictions on the use of the ALTER INDEX statement

Rebuilding an Existing Index

Before rebuilding an existing index, compare the costs and benefits associated with rebuilding to those associated with coalescing indexes as described in Table 15–1 on page 15-8.

When you rebuild an index, you use an existing index as the data source. Creating an index in this manner enables you to change storage characteristics or move to a new tablespace. Rebuilding an index based on an existing data source removes intra-block fragmentation. Compared to dropping the index and using the CREATE INDEX statement, re-creating an existing index offers better performance.

The following statement rebuilds the existing index emp_name:

```
ALTER INDEX emp_name REBUILD;
```

The REBUILD clause must immediately follow the index name, and precede any other options. It cannot be used in conjunction with the DEALLOCATE UNUSED clause.

You have the option of rebuilding the index online. The following statement rebuilds the emp_name index online:

```
ALTER INDEX emp_name REBUILD ONLINE;
```

If you do not have the space required to rebuild an index, you can choose instead to coalesce the index. Coalescing an index is an online operation.

See Also:

- "Creating an Index Online" on page 15-13
- "Monitoring Space Use of Indexes" on page 15-19

Monitoring Index Usage

Oracle Database provides a means of monitoring indexes to determine whether they are being used. If an index is not being used, then it can be dropped, eliminating unnecessary statement overhead.

To start monitoring the usage of an index, issue this statement:

```
ALTER INDEX index MONITORING USAGE;
```

Later, issue the following statement to stop the monitoring:

```
ALTER INDEX index NOMONITORING USAGE;
```

The view V$OBJECT_USAGE can be queried for the index being monitored to see if the index has been used. The view contains a USED column whose value is YES or NO, depending upon if the index has been used within the time period being monitored. The view also contains the start and stop times of the monitoring period, and a MONITORING column (YES/NO) to indicate if usage monitoring is currently active.

Each time that you specify MONITORING USAGE, the V$OBJECT_USAGE view is reset for the specified index. The previous usage information is cleared or reset, and a new start time is recorded. When you specify NOMONITORING USAGE, no further monitoring is performed, and the end time is recorded for the monitoring period. Until the next ALTER INDEX ... MONITORING USAGE statement is issued, the view information is left unchanged.

Monitoring Space Use of Indexes

If key values in an index are inserted, updated, and deleted frequently, the index can lose its acquired space efficiently over time. Monitor index efficiency of space usage at regular intervals by first analyzing the index structure, using the ANALYZE INDEX ... VALIDATE STRUCTURE statement, and then querying the INDEX_STATS view:

```
SELECT PCT_USED FROM INDEX_STATS WHERE NAME = 'index';
```

The percentage of index space usage varies according to how often index keys are inserted, updated, or deleted. Develop a history of average efficiency of space usage for an index by performing the following sequence of operations several times:

- Analyzing statistics
- Validating the index
- Checking PCT_USED
- Dropping and rebuilding (or coalescing) the index

When you find that index space usage drops below its average, you can condense the index space by dropping the index and rebuilding it, or coalescing it.

See Also: "Analyzing Tables, Indexes, and Clusters" on page 20-2

Dropping Indexes

To drop an index, the index must be contained in your schema, or you must have the DROP ANY INDEX system privilege.

Some reasons for dropping an index include:

- The index is no longer required.
- The index is not providing anticipated performance improvements for queries issued against the associated table. For example, the table might be very small, or there might be many rows in the table but very few index entries.
- Applications do not use the index to query the data.
- The index has become invalid and must be dropped before being rebuilt.
- The index has become too fragmented and must be dropped before being rebuilt.

When you drop an index, all extents of the index segment are returned to the containing tablespace and become available for other objects in the tablespace.

How you drop an index depends on whether you created the index explicitly with a CREATE INDEX statement, or implicitly by defining a key constraint on a table. If you created the index explicitly with the CREATE INDEX statement, then you can drop the index with the DROP INDEX statement. The following statement drops the emp_ename index:

```
DROP INDEX emp_ename;
```

You cannot drop only the index associated with an enabled UNIQUE key or PRIMARY KEY constraint. To drop a constraints associated index, you must disable or drop the constraint itself.

Note: If a table is dropped, all associated indexes are dropped automatically.

See Also:

- *Oracle Database SQL Reference* for syntax and restrictions on the use of the DROP INDEX statement
- "Managing Integrity Constraints" on page 20-12

Viewing Index Information

The following views display information about indexes:

View	Description
DBA_INDEXES ALL_INDEXES USER_INDEXES	DBA view describes indexes on all tables in the database. ALL view describes indexes on all tables accessible to the user. USER view is restricted to indexes owned by the user. Some columns in these views contain statistics that are generated by the DBMS_STATS package or ANALYZE statement.
DBA_IND_COLUMNS ALL_IND_COLUMNS USER_IND_COLUMNS	These views describe the columns of indexes on tables. Some columns in these views contain statistics that are generated by the DBMS_STATS package or ANALYZE statement.
DBA_IND_EXPRESSIONS ALL_IND_EXPRESSIONS USER_IND_EXPRESSIONS	These views describe the expressions of function-based indexes on tables.
INDEX_STATS	Stores information from the last ANALYZE INDEX ... VALIDATE STRUCTURE statement.
INDEX_HISTOGRAM	Stores information from the last ANALYZE INDEX ... VALIDATE STRUCTURE statement.
V$OBJECT_USAGE	Contains index usage information produced by the ALTER INDEX ... MONITORING USAGE functionality.

See Also: *Oracle Database Reference* for a complete description of these views

16

Managing Partitioned Tables and Indexes

This chapter describes various aspects of managing partitioned tables and indexes, and contains the following topics:

About Partitioned Tables and Indexes

Modern enterprises frequently run mission-critical databases containing upwards of several hundred gigabytes and, in many cases, several terabytes of data. These enterprises are challenged by the support and maintenance requirements of very large databases (VLDB), and must devise methods to meet those challenges.

One way to meet VLDB demands is to create and use **partitioned tables and indexes**. Partitioned tables allow your data to be broken down into smaller, more manageable pieces called **partitions**, or even **subpartitions**. Indexes can be partitioned in similar fashion. Each partition is stored in its own segment and can be managed individually. It can function independently of the other partitions, thus providing a structure that can be better tuned for availability and performance.

If you are using parallel execution, partitions provide another means of parallelization. Operations on partitioned tables and indexes are performed in parallel by assigning different parallel execution servers to different partitions of the table or index.

Partitions and subpartitions of a table or index all share the same logical attributes. For example, all partitions (or subpartitions) in a table share the same column and constraint definitions, and all partitions (or subpartitions) of an index share the same index options. They can, however, have different physical attributes (such as TABLESPACE).

Although you are not required to keep each table or index partition (or subpartition) in a separate tablespace, it is to your advantage to do so. Storing partitions in separate tablespaces enables you to:

- Reduce the possibility of data corruption in multiple partitions
- Back up and recover each partition independently
- Control the mapping of partitions to disk drives (important for balancing I/O load)
- Improve manageability, availability, and performance

Partitioning is transparent to existing applications and standard DML statements run against partitioned tables. However, an application can be programmed to take advantage of partitioning by using partition-extended table or index names in DML.

You can use SQL*Loader and the import and export utilities to load or unload data stored in partitioned tables. These utilities are all partition and subpartition aware.

See Also:

- Chapter 13, "Managing Space for Schema Objects" is recommended reading before attempting tasks described in this chapter.
- *Oracle Database Concepts* contains more information about partitioning. Before the first time you attempt to create a partitioned table or index, or perform maintenance operations on any partitioned table, it is recommended that you review the information contained in that book.
- *Oracle Data Warehousing Guide* and *Oracle Database Concepts* contain information about parallel execution
- *Oracle Database Utilities* describes SQL*Loader and the import and export utilities.

Partitioning Methods

There are several partitioning methods offered by Oracle Database:

- Range partitioning
- Hash partitioning
- List partitioning
- Composite range-hash partitioning
- Composite range-list partitioning

Indexes, as well as tables, can be partitioned. A global index can be partitioned by the range or hash method, and it can be defined on any type of partitioned, or nonpartitioned, table. It can require more maintenance than a local index.

A local index is constructed so that it reflects the structure of the underlying table. It is equipartitioned with the underlying table, meaning that it is partitioned on the same columns as the underlying table, creates the same number of partitions or subpartitions, and gives them the same partition bounds as corresponding partitions of the underlying table. For local indexes, index partitioning is maintained automatically when partitions are affected by maintenance activity. This ensures that the index remains equipartitioned with the underlying table.

The following sections can help you decide on a partitioning method appropriate for your needs:

- When to Use Range Partitioning
- When to Use Hash Partitioning
- When to Use List Partitioning
- When to Use Composite Range-Hash Partitioning
- When to Use Composite Range-List Partitioning

When to Use Range Partitioning

Use range partitioning to map rows to partitions based on ranges of column values. This type of partitioning is useful when dealing with data that has logical ranges into which it can be distributed; for example, months of the year. Performance is best when the data evenly distributes across the range. If partitioning by range causes partitions to vary dramatically in size because of unequal distribution, you may want to consider one of the other methods of partitioning.

When creating range partitions, you must specify:

- Partitioning method: range
- Partitioning column(s)
- Partition descriptions identifying partition bounds

The example below creates a table of four partitions, one for each quarter of sales. The columns `sale_year`, `sale_month`, and `sale_day` are the **partitioning columns**, while their values constitute the **partitioning key** of a specific row. The `VALUES LESS THAN` clause determines the **partition bound**: rows with partitioning key values that compare less than the ordered list of values specified by the clause are stored in the partition. Each partition is given a name (`sales_q1`, `sales_q2`, ...), and each partition is contained in a separate tablespace (`tsa`, `tsb`, ...).

```
CREATE TABLE sales
    ( invoice_no NUMBER,
      sale_year  INT NOT NULL,
      sale_month INT NOT NULL,
      sale_day   INT NOT NULL )
  PARTITION BY RANGE (sale_year, sale_month, sale_day)
    ( PARTITION sales_q1 VALUES LESS THAN (1999, 04, 01)
        TABLESPACE tsa,
      PARTITION sales_q2 VALUES LESS THAN (1999, 07, 01)
        TABLESPACE tsb,
      PARTITION sales_q3 VALUES LESS THAN (1999, 10, 01)
        TABLESPACE tsc,
      PARTITION sales_q4 VALUES LESS THAN (2000, 01, 01)
        TABLESPACE tsd );
```

A row with `sale_year=1999`, `sale_month=8`, and `sale_day=1` has a partitioning key of `(1999, 8, 1)` and would be stored in partition `sales_q3`.

See Also: "Using Multicolumn Partitioning Keys" on page 16-20

When to Use Hash Partitioning

Use hash partitioning if your data does not easily lend itself to range partitioning, but you would like to partition for performance and manageability reasons. Hash partitioning provides a method of evenly distributing data across a specified number of partitions. Rows are mapped into partitions based on a hash value of the partitioning key. Creating and using hash partitions gives you a highly tunable

method of data placement, because you can influence availability and performance by spreading these evenly sized partitions across I/O devices (striping).

To create hash partitions you specify the following:

- Partitioning method: hash
- Partitioning column(s)
- Number of partitions or individual partition descriptions

The following example creates a hash-partitioned table. The partitioning column is `id`, four partitions are created and assigned system generated names, and they are placed in four named tablespaces (`gear1`, `gear2`, ...).

```
CREATE TABLE scubagear
     (id NUMBER,
      name VARCHAR2 (60))
   PARTITION BY HASH (id)
   PARTITIONS 4
   STORE IN (gear1, gear2, gear3, gear4);
```

See Also: "Using Multicolumn Partitioning Keys" on page 16-20

When to Use List Partitioning

Use list partitioning when you require explicit control over how rows map to partitions. You can specify a list of discrete values for the partitioning column in the description for each partition. This is different from range partitioning, where a range of values is associated with a partition, and from hash partitioning, where the user has no control of the row to partition mapping.

The list partitioning method is specifically designed for modeling data distributions that follow discrete values. This cannot be easily done by range or hash partitioning because:

- Range partitioning assumes a natural range of values for the partitioning column. It is not possible to group together out-of-range values partitions.
- Hash partitioning allows no control over the distribution of data because the data is distributed over the various partitions using the system hash function. Again, this makes it impossible to logically group together discrete values for the partitioning columns into partitions.

Further, list partitioning allows unordered and unrelated sets of data to be grouped and organized together very naturally.

Unlike the range and hash partitioning methods, multicolumn partitioning is not supported for list partitioning. If a table is partitioned by list, the partitioning key can consist only of a single column of the table. Otherwise all columns that can be partitioned by the range or hash methods can be partitioned by the list partitioning method.

When creating list partitions, you must specify:

- Partitioning method: list
- Partitioning column
- Partition descriptions, each specifying a list of literal values (a **value list**), which are the discrete values of the partitioning column that qualify a row to be included in the partition

The following example creates a list-partitioned table. It creates table `q1_sales_by_region` which is partitioned by regions consisting of groups of states.

```
CREATE TABLE q1_sales_by_region
      (deptno number,
       deptname varchar2(20),
       quarterly_sales number(10, 2),
       state varchar2(2))
   PARTITION BY LIST (state)
      (PARTITION q1_northwest VALUES ('OR', 'WA'),
       PARTITION q1_southwest VALUES ('AZ', 'UT', 'NM'),
       PARTITION q1_northeast VALUES  ('NY', 'VM', 'NJ'),
       PARTITION q1_southeast VALUES ('FL', 'GA'),
       PARTITION q1_northcentral VALUES ('SD', 'WI'),
       PARTITION q1_southcentral VALUES ('OK', 'TX'));
```

A row is mapped to a partition by checking whether the value of the partitioning column for a row matches a value in the value list that describes the partition.

For example, some sample rows are inserted as follows:

- (10, 'accounting', 100, 'WA') maps to partition `q1_northwest`
- (20, 'R&D', 150, 'OR') maps to partition `q1_northwest`
- (30, 'sales', 100, 'FL') maps to partition `q1_southeast`
- (40, 'HR', 10, 'TX') maps to partition `q1_southwest`
- (50, 'systems engineering', 10, 'CA') does not map to any partition in the table and raises an error

Unlike range partitioning, with list partitioning, there is no apparent sense of order between partitions. You can also specify a **default partition** into which rows that do not map to any other partition are mapped. If a default partition were specified in the preceding example, the state CA would map to that partition.

When to Use Composite Range-Hash Partitioning

Range-hash partitioning partitions data using the range method, and within each partition, subpartitions it using the hash method. These composite partitions are ideal for both historical data and striping, and provide improved manageability of range partitioning and data placement, as well as the parallelism advantages of hash partitioning.

When creating range-hash partitions, you specify the following:

- Partitioning method: range
- Partitioning column(s)
- Partition descriptions identifying partition bounds
- Subpartitioning method: hash
- Subpartitioning column(s)
- Number of subpartitions for each partition or descriptions of subpartitions

The following statement creates a range-hash partitioned table. In this example, three range partitions are created, each containing eight subpartitions. Because the subpartitions are not named, system generated names are assigned, but the `STORE IN` clause distributes them across the 4 specified tablespaces (`ts1`, ...,`ts4`).

```
CREATE TABLE scubagear (equipno NUMBER, equipname VARCHAR(32), price NUMBER)
  PARTITION BY RANGE (equipno) SUBPARTITION BY HASH(equipname)
    SUBPARTITIONS 8 STORE IN (ts1, ts2, ts3, ts4)
      (PARTITION p1 VALUES LESS THAN (1000),
       PARTITION p2 VALUES LESS THAN (2000),
       PARTITION p3 VALUES LESS THAN (MAXVALUE));
```

The partitions of a range-hash partitioned table are logical structures only, as their data is stored in the segments of their subpartitions. As with partitions, these subpartitions share the same logical attributes. Unlike range partitions in a range-partitioned table, the subpartitions cannot have different physical attributes from the owning partition, although they are not required to reside in the same tablespace.

When to Use Composite Range-List Partitioning

Like the composite range-hash partitioning method, the composite range-list partitioning method provides for partitioning based on a two level hierarchy. The first level of partitioning is based on a range of values, as for range partitioning; the second level is based on discrete values, as for list partitioning. This form of composite partitioning is well suited for historical data, but lets you further group the rows of data based on unordered or unrelated column values.

When creating range-list partitions, you specify the following:

- Partitioning method: range
- Partitioning column(s)
- Partition descriptions identifying partition bounds
- Subpartitioning method: list
- Subpartitioning column
- Subpartition descriptions, each specifying a list of literal values (a **value list**), which are the discrete values of the subpartitioning column that qualify a row to be included in the subpartition

The following example illustrates how range-list partitioning might be used. The example tracks sales data of products by quarters and within each quarter, groups it by specified states.

```
CREATE TABLE quarterly_regional_sales
      (deptno number, item_no varchar2(20),
       txn_date date, txn_amount number, state varchar2(2))
  TABLESPACE ts4
  PARTITION BY RANGE (txn_date)
    SUBPARTITION BY LIST (state)
      (PARTITION q1_1999 VALUES LESS THAN (TO_DATE('1-APR-1999','DD-MON-YYYY'))
         (SUBPARTITION q1_1999_northwest VALUES ('OR', 'WA'),
          SUBPARTITION q1_1999_southwest VALUES ('AZ', 'UT', 'NM'),
          SUBPARTITION q1_1999_northeast VALUES ('NY', 'VM', 'NJ'),
          SUBPARTITION q1_1999_southeast VALUES ('FL', 'GA'),
          SUBPARTITION q1_1999_northcentral VALUES ('SD', 'WI'),
          SUBPARTITION q1_1999_southcentral VALUES ('OK', 'TX')
         ),
       PARTITION q2_1999 VALUES LESS THAN ( TO_DATE('1-JUL-1999','DD-MON-YYYY'))
         (SUBPARTITION q2_1999_northwest VALUES ('OR', 'WA'),
          SUBPARTITION q2_1999_southwest VALUES ('AZ', 'UT', 'NM'),
          SUBPARTITION q2_1999_northeast VALUES ('NY', 'VM', 'NJ'),
          SUBPARTITION q2_1999_southeast VALUES ('FL', 'GA'),
```

```
    SUBPARTITION q2_1999_northcentral VALUES ('SD', 'WI'),
    SUBPARTITION q2_1999_southcentral VALUES ('OK', 'TX')
   ),
  PARTITION q3_1999 VALUES LESS THAN (TO_DATE('1-OCT-1999','DD-MON-YYYY'))
   (SUBPARTITION q3_1999_northwest VALUES ('OR', 'WA'),
    SUBPARTITION q3_1999_southwest VALUES ('AZ', 'UT', 'NM'),
    SUBPARTITION q3_1999_northeast VALUES ('NY', 'VM', 'NJ'),
    SUBPARTITION q3_1999_southeast VALUES ('FL', 'GA'),
    SUBPARTITION q3_1999_northcentral VALUES ('SD', 'WI'),
    SUBPARTITION q3_1999_southcentral VALUES ('OK', 'TX')
   ),
  PARTITION q4_1999 VALUES LESS THAN ( TO_DATE('1-JAN-2000','DD-MON-YYYY'))
   (SUBPARTITION q4_1999_northwest VALUES ('OR', 'WA'),
    SUBPARTITION q4_1999_southwest VALUES ('AZ', 'UT', 'NM'),
    SUBPARTITION q4_1999_northeast VALUES ('NY', 'VM', 'NJ'),
    SUBPARTITION q4_1999_southeast VALUES ('FL', 'GA'),
    SUBPARTITION q4_1999_northcentral VALUES ('SD', 'WI'),
    SUBPARTITION q4_1999_southcentral VALUES ('OK', 'TX')
   )
);
```

A row is mapped to a partition by checking whether the value of the partitioning column for a row falls within a specific partition range. The row is then mapped to a subpartition within that partition by identifying the subpartition whose descriptor value list contains a value matching the subpartition column value.

For example, some sample rows are inserted as follows:

- (10, 4532130, '23-Jan-1999', 8934.10, 'WA') maps to subpartition `q1_1999_northwest`
- (20, 5671621, '15-May-1999', 49021.21, 'OR') maps to subpartition `q2_1999_northwest`
- (30, 9977612, ,'07-Sep-1999', 30987.90, 'FL') maps to subpartition `q3_1999_southeast`
- (40, 9977612, '29-Nov-1999', 67891.45, 'TX') maps to subpartition `q4_1999_southcentral`
- (40, 4532130, '5-Jan-2000', 897231.55, 'TX') does not map to any partition in the table and raises an error
- (50, 5671621, '17-Dec-1999', 76123.35, 'CA') does not map to any subpartition in the table and raises an error

The partitions of a range-list partitioned table are logical structures only, as their data is stored in the segments of their subpartitions. The list subpartitions have the

same characteristics as list partitions. You can specify a default subpartition, just as you specify a default partition for list partitioning.

Creating Partitioned Tables

Creating a partitioned table or index is very similar to creating a nonpartitioned table or index (as described in Chapter 14, "Managing Tables"), but you include a partitioning clause. The partitioning clause, and subclauses, that you include depend upon the type of partitioning you want to achieve.

You can partition both regular (heap organized) tables and index-organized tables, except for those containing `LONG` or `LONG RAW` columns. You can create nonpartitioned global indexes, range or hash-partitioned global indexes, and local indexes on partitioned tables.

When you create (or alter) a partitioned table, a row movement clause, either `ENABLE ROW MOVEMENT` or `DISABLE ROW MOVEMENT` can be specified. This clause either enables or disables the migration of a row to a new partition if its key is updated. The default is `DISABLE ROW MOVEMENT`.

The following sections present details and examples of creating partitions for the various types of partitioned tables and indexes:

- Creating Range-Partitioned Tables and Global Indexes
- Creating Hash-Partitioned Tables and Global Indexes
- Creating List-Partitioned Tables
- Creating Composite Range-Hash Partitioned Tables
- Creating Composite Range-List Partitioned Tables
- Using Subpartition Templates to Describe Composite Partitioned Tables
- Using Multicolumn Partitioning Keys
- Creating Partitioned Index-Organized Tables
- Partitioning Restrictions for Multiple Block Sizes

See Also:

- *Oracle Database SQL Reference* for the exact syntax of the partitioning clauses for creating and altering partitioned tables and indexes, any restrictions on their use, and specific privileges required for creating and altering tables
- *Oracle Database Application Developer's Guide - Large Objects* for information specific to creating partitioned tables containing columns with LOBs or other objects stored as LOBs
- *Oracle Database Application Developer's Guide - Object-Relational Features* for information specific to creating tables with object types, nested tables, or VARRAYs.

Creating Range-Partitioned Tables and Global Indexes

The PARTITION BY RANGE clause of the CREATE TABLE statement specifies that the table or index is to be range-partitioned. The PARTITION clauses identify the individual partition ranges, and optional subclauses of a PARTITION clause can specify physical and other attributes specific to a partition segment. If not overridden at the partition level, partitions inherit the attributes of their underlying table.

Creating a Range Partitioned Table

In this example, more complexity is added to the example presented earlier for a range-partitioned table. Storage parameters and a LOGGING attribute are specified at the table level. These replace the corresponding defaults inherited from the tablespace level for the table itself, and are inherited by the range partitions. However, since there was little business in the first quarter, the storage attributes for partition sales_q1 are made smaller. The ENABLE ROW MOVEMENT clause is specified to allow the migration of a row to a new partition if an update to a key value is made that would place the row in a different partition.

```
CREATE TABLE sales
     ( invoice_no NUMBER,
       sale_year  INT NOT NULL,
       sale_month INT NOT NULL,
       sale_day   INT NOT NULL )
   STORAGE (INITIAL 100K NEXT 50K) LOGGING
   PARTITION BY RANGE ( sale_year, sale_month, sale_day)
     ( PARTITION sales_q1 VALUES LESS THAN ( 1999, 04, 01 )
        TABLESPACE tsa STORAGE (INITIAL 20K, NEXT 10K),
```

```
      PARTITION sales_q2 VALUES LESS THAN ( 1999, 07, 01 )
       TABLESPACE tsb,
      PARTITION sales_q3 VALUES LESS THAN ( 1999, 10, 01 )
       TABLESPACE tsc,
      PARTITION sales q4 VALUES LESS THAN ( 2000, 01, 01 )
       TABLESPACE tsd)
   ENABLE ROW MOVEMENT;
```

Creating a Range-Partitioned Global Index

The rules for creating range-partitioned global indexes are similar to those for creating range-partitioned tables. The following is an example of creating a range-partitioned global index on `sales_month` for the table created in the preceding example. Each index partition is named but is stored in the default tablespace for the index.

```
CREATE INDEX month_ix ON sales(sales_month)
   GLOBAL PARTITION BY RANGE(sales_month)
      (PARTITION pm1_ix VALUES LESS THAN (2)
       PARTITION pm2_ix VALUES LESS THAN (3)
       PARTITION pm3_ix VALUES LESS THAN (4)
       PARTITION pm4_ix VALUES LESS THAN (5)
       PARTITION pm5_ix VALUES LESS THAN (6)
       PARTITION pm6_ix VALUES LESS THAN (7)
       PARTITION pm7_ix VALUES LESS THAN (8)
       PARTITION pm8_ix VALUES LESS THAN (9)
       PARTITION pm9_ix VALUES LESS THAN (10)
       PARTITION pm10_ix VALUES LESS THAN (11)
       PARTITION pm11_ix VALUES LESS THAN (12)
       PARTITION pm12_ix VALUES LESS THAN (MAXVALUE));
```

Note: If your enterprise has or will have databases using different character sets, use caution when partitioning on character columns, because the sort sequence of characters is not identical in all character sets. For more information, see *Oracle Database Globalization Support Guide.*

Creating Hash-Partitioned Tables and Global Indexes

The `PARTITION BY HASH` clause of the `CREATE TABLE` statement identifies that the table is to be hash-partitioned. The `PARTITIONS` clause can then be used to specify the number of partitions to create, and optionally, the tablespaces to store

them in. Alternatively, you can use PARTITION clauses to name the individual partitions and their tablespaces.

The only attribute you can specify for hash partitions is TABLESPACE. All of the hash partitions of a table must share the same segment attributes (except TABLESPACE), which are inherited from the table level.

Creating a Hash Partitioned Table

The following examples illustrate two methods of creating a hash-partitioned table named dept. In the first example the number of partitions is specified, but system generated names are assigned to them and they are stored in the default tablespace of the table.

```
CREATE TABLE dept (deptno NUMBER, deptname VARCHAR(32))
     PARTITION BY HASH(deptno) PARTITIONS 16;
```

In this second example, names of individual partitions, and tablespaces in which they are to reside, are specified. The initial extent size for each hash partition (segment) is also explicitly stated at the table level, and all partitions inherit this attribute.

```
CREATE TABLE dept (deptno NUMBER, deptname VARCHAR(32))
     STORAGE (INITIAL 10K)
     PARTITION BY HASH(deptno)
       (PARTITION p1 TABLESPACE ts1, PARTITION p2 TABLESPACE ts2,
        PARTITION p3 TABLESPACE ts1, PARTITION p4 TABLESPACE ts3);
```

If you create a local index for this table, the database constructs the index so that it is equipartitioned with the underlying table. The database also ensures that the index is maintained automatically when maintenance operations are performed on the underlying table. The following is an example of creating a local index on the table dept:

```
CREATE INDEX loc_dept_ix ON dept(deptno) LOCAL;
```

You can optionally name the hash partitions and tablespaces into which the local index partitions are to be stored, but if you do not do so, the database uses the name of the corresponding base partition as the index partition name, and stores the index partition in the same tablespace as the table partition.

Creating a Hash-Partitioned Global Index

Hash partitioned global indexes can improve the performance of indexes where a small number of leaf blocks in the index have high contention in multiuser OLTP

environments. Queries involving the equality and IN predicates on the index partitioning key can efficiently use hash-partitioned global indexes.

The syntax for creating a hash partitioned global index is similar to that used for a hash partitioned table. For example, the following statement creates a hash-partitioned global index:

```
CREATE INDEX hgidx ON tab (c1,c2,c3) GLOBAL
     PARTITION BY HASH (c1,c2)
     (PARTITION p1  TABLESPACE tbs_1,
      PARTITION p2  TABLESPACE tbs_2,
      PARTITION p3  TABLESPACE tbs_3,
      PARTITION p4  TABLESPACE tbs_4);
```

Creating List-Partitioned Tables

The semantics for creating list partitions are very similar to those for creating range partitions. However, to create list partitions, you specify a PARTITION BY LIST clause in the CREATE TABLE statement, and the PARTITION clauses specify lists of literal values, which are the discrete values of the partitioning columns that qualify rows to be included in the partition. For list partitioning, the partitioning key can only be a single column name from the table.

Available only with list partitioning, you can use the keyword DEFAULT to describe the value list for a partition. This identifies a partition that will accommodate rows that do not map into any of the other partitions.

As for range partitions, optional subclauses of a PARTITION clause can specify physical and other attributes specific to a partition segment. If not overridden at the partition level, partitions inherit the attributes of their parent table.

The following example creates table sales_by_region and partitions it using the list method. The first two PARTITION clauses specify physical attributes, which override the table-level defaults. The remaining PARTITION clauses do not specify attributes and those partitions inherit their physical attributes from table-level defaults. A default partition is specified.

```
CREATE TABLE sales_by_region (item# INTEGER, qty INTEGER,
             store_name VARCHAR(30), state_code VARCHAR(2),
             sale_date DATE)
     STORAGE(INITIAL 10K NEXT 20K) TABLESPACE tbs5
     PARTITION BY LIST (state_code)
     (
     PARTITION region_east
        VALUES ('MA','NY','CT','NH','ME','MD','VA','PA','NJ')
```

```
   STORAGE (INITIAL 20K NEXT 40K PCTINCREASE 50)
   TABLESPACE tbs8,
PARTITION region_west
   VALUES ('CA','AZ','NM','OR','WA','UT','NV','CO')
   PCTFREE 25 NOLOGGING,
PARTITION region_south
   VALUES ('TX','KY','TN','LA','MS','AR','AL','GA'),
PARTITION region_central
   VALUES ('OH','ND','SD','MO','IL','MI','IA'),
PARTITION region_null
   VALUES (NULL),
PARTITION region_unknown
   VALUES (DEFAULT)
);
```

Creating Composite Range-Hash Partitioned Tables

To create a range-hash partitioned table, you start by using the PARTITION BY RANGE clause of a CREATE TABLE statement. Next, you specify a SUBPARTITION BY HASH clause that follows similar syntax and rules as the PARTITION BY HASH clause. The individual PARTITION and SUBPARTITION or SUBPARTITIONS clauses, and optionally a SUBPARTITION TEMPLATE clause, follow.

Attributes specified for a range partition apply to all subpartitions of that partition. You can specify different attributes for each range partition, and you can specify a STORE IN clause at the partition level if the list of tablespaces across which the subpartitions of that partition should be spread is different from those of other partitions. All of this is illustrated in the following example.

```
CREATE TABLE emp (deptno NUMBER, empname VARCHAR(32), grade NUMBER)
     PARTITION BY RANGE(deptno) SUBPARTITION BY HASH(empname)
        SUBPARTITIONS 8 STORE IN (ts1, ts3, ts5, ts7)
    (PARTITION p1 VALUES LESS THAN (1000) PCTFREE 40,
     PARTITION p2 VALUES LESS THAN (2000)
        STORE IN (ts2, ts4, ts6, ts8),
     PARTITION p3 VALUES LESS THAN (MAXVALUE)
       (SUBPARTITION p3_s1 TABLESPACE ts4,
        SUBPARTITION p3_s2 TABLESPACE ts5));
```

To learn how using a subpartition template can simplify the specification of a composite partitioned table, see "Using Subpartition Templates to Describe Composite Partitioned Tables" on page 16-18.

The following statement is an example of creating a local index on the emp table where the index segments are spread across tablespaces ts7, ts8, and ts9.

```
CREATE INDEX emp_ix ON emp(deptno)
     LOCAL STORE IN (ts7, ts8, ts9);
```

This local index is equipartitioned with the base table as follows:

- It consists of as many partitions as the base table.
- Each index partition consists of as many subpartitions as the corresponding base table partition.
- Index entries for rows in a given subpartition of the base table are stored in the corresponding subpartition of the index.

Creating Composite Range-List Partitioned Tables

The concept of range-list partitioning is similar to that of the other composite partitioning method, range-hash, but this time you specify that the subpartitions are to be list rather than hash. Specifically, after the CREATE TABLE ... PARTITION BY RANGE clause, you include a SUBPARTITION BY LIST clause that follows similar syntax and rules as the PARTITION BY LIST clause. The individual PARTITION and SUBPARTITION clauses, and optionally a SUBPARTITION TEMPLATE clause, follow.

The range partitions of the composite partitioned table are described as for noncomposite range partitioned tables. This allows that optional subclauses of a PARTITION clause can specify physical and other attributes, including tablespace, specific to a partition segment. If not overridden at the partition level, partitions inherit the attributes of their underlying table.

The list subpartition descriptions, in the SUBPARTITION clauses, are described as for noncomposite list partitions, except the only physical attribute that can be specified is a tablespace (optional). Subpartitions inherit all other physical attributes from the partition description.

The following example of creates a table that specifies a tablespace at the partition and subpartition levels. The number of subpartitions within each partition varies, and default subpartitions are specified.

```
CREATE TABLE sample_regional_sales
      (deptno number, item_no varchar2(20),
       txn_date date, txn_amount number, state varchar2(2))
  PARTITION BY RANGE (txn_date)
    SUBPARTITION BY LIST (state)
      (PARTITION q1_1999 VALUES LESS THAN (TO_DATE('1-APR-1999','DD-MON-YYYY'))
          TABLESPACE tbs_1
        (SUBPARTITION q1_1999_northwest VALUES ('OR', 'WA'),
```

```
   SUBPARTITION q1_1999_southwest VALUES ('AZ', 'UT', 'NM'),
   SUBPARTITION q1_1999_northeast VALUES ('NY', 'VM', 'NJ'),
   SUBPARTITION q1_1999_southeast VALUES ('FL', 'GA'),
   SUBPARTITION q1_others VALUES (DEFAULT) TABLESPACE tbs_4
  ),
 PARTITION q2_1999 VALUES LESS THAN ( TO_DATE('1-JUL-1999','DD-MON-YYYY'))
   TABLESPACE tbs_2
  (SUBPARTITION q2_1999_northwest VALUES ('OR', 'WA'),
   SUBPARTITION q2_1999_southwest VALUES ('AZ', 'UT', 'NM'),
   SUBPARTITION q2_1999_northeast VALUES ('NY', 'VM', 'NJ'),
   SUBPARTITION q2_1999_southeast VALUES ('FL', 'GA'),
   SUBPARTITION q2_1999_northcentral VALUES ('SD', 'WI'),
   SUBPARTITION q2_1999_southcentral VALUES ('OK', 'TX')
  ),
 PARTITION q3_1999 VALUES LESS THAN (TO_DATE('1-OCT-1999','DD-MON-YYYY'))
   TABLESPACE tbs_3
  (SUBPARTITION q3_1999_northwest VALUES ('OR', 'WA'),
   SUBPARTITION q3_1999_southwest VALUES ('AZ', 'UT', 'NM'),
   SUBPARTITION q3_others VALUES (DEFAULT) TABLESPACE tbs_4
  ),
 PARTITION q4_1999 VALUES LESS THAN ( TO_DATE('1-JAN-2000','DD-MON-YYYY'))
   TABLESPACE tbs_4
);
```

This example results in the following subpartition descriptions:

- All subpartitions inherit their physical attributes, other than tablespace, from tablespace level defaults. This is because the only physical attribute that has been specified for partitions or subpartitions is tablespace. There are no table level physical attributes specified, thus tablespace level defaults are inherited at all levels.
- The first 4 subpartitions of partition `q1_1999` are all contained in `tbs_1`, except for the subpartition `q1_others`, which is stored in `tbs_4` and contains all rows that do not map to any of the other partitions.
- The 6 subpartitions of partition `q2_1999` are all stored in `tbs_2`.
- The first 2 subpartitions of partition `q3_1999` are all contained in `tbs_3`, except for the subpartition `q3_others`, which is stored in `tbs_4` and contains all rows that do not map to any of the other partitions.
- There is no subpartition description for partition `q4_1999`. This results in one default subpartition being created and stored in `tbs_4`. The subpartition name is system generated in the form SYS_SUBP*n*.

To learn how using a subpartition template can simplify the specification of a composite partitioned table, see "Using Subpartition Templates to Describe Composite Partitioned Tables".

Using Subpartition Templates to Describe Composite Partitioned Tables

You can create subpartitions in a composite partitioned table using a subpartition template. A subpartition template simplifies the specification of subpartitions by not requiring that a subpartition descriptor be specified for every partition in the table. Instead, you describe subpartitions only once in a template, then apply that subpartition template to every partition in the table.

The subpartition template is used whenever a subpartition descriptor is not specified for a partition. If a subpartition descriptor is specified, then it is used instead of the subpartition template for that partition. If no subpartition template is specified, and no subpartition descriptor is supplied for a partition, then a single default subpartition is created.

Specifying a Subpartition Template for a Range-Hash Partitioned Table

In the case of range-hash partitioned tables, the subpartition template can describe the subpartitions in detail, or it can specify just the number of hash subpartitions.

The following example creates a range-hash partitioned table using a subpartition template:

```
CREATE TABLE emp_sub_template (deptno NUMBER, empname VARCHAR(32), grade NUMBER)
     PARTITION BY RANGE(deptno) SUBPARTITION BY HASH(empname)
     SUBPARTITION TEMPLATE
         (SUBPARTITION a TABLESPACE ts1,
          SUBPARTITION b TABLESPACE ts2,
          SUBPARTITION c TABLESPACE ts3,
          SUBPARTITION d TABLESPACE ts4
         )
    (PARTITION p1 VALUES LESS THAN (1000),
     PARTITION p2 VALUES LESS THAN (2000),
     PARTITION p3 VALUES LESS THAN (MAXVALUE)
    );
```

This example produces the following table description:

- Every partition has four subpartitions as described in the subpartition template.

- Each subpartition has a tablespace specified. It is required that if a tablespace is specified for one subpartition in a subpartition template, then one must be specified for all.
- The names of the subpartitions are generated by concatenating the partition name with the subpartition name in the form:

  partition name_subpartition name

The following query displays the subpartition names and tablespaces:

```
SQL> SELECT TABLESPACE_NAME, PARTITION_NAME, SUBPARTITION_NAME
  2  FROM DBA_TAB_SUBPARTITIONS WHERE TABLE_NAME='EMP_SUB_TEMPLATE'
  3  ORDER BY TABLESPACE_NAME;

TABLESPACE_NAME PARTITION_NAME  SUBPARTITION_NAME
--------------- --------------- ------------------
TS1             P1              P1_A
TS1             P2              P2_A
TS1             P3              P3_A
TS2             P1              P1_B
TS2             P2              P2_B
TS2             P3              P3_B
TS3             P1              P1_C
TS3             P2              P2_C
TS3             P3              P3_C
TS4             P1              P1_D
TS4             P2              P2_D
TS4             P3              P3_D

12 rows selected.
```

Specifying a Subpartition Template for a Range-List Partitioned Table

The following example, for a range-list partitioned table, illustrates how using a subpartition template can help you stripe data across tablespaces. In this example a table is created where the table subpartitions are vertically striped, meaning that subpartition *n* from every partition is in the same tablespace.

```
CREATE TABLE stripe_regional_sales
            ( deptno number, item_no varchar2(20),
              txn_date date, txn_amount number, state varchar2(2))
   PARTITION BY RANGE (txn_date)
   SUBPARTITION BY LIST (state)
   SUBPARTITION TEMPLATE
      (SUBPARTITION northwest VALUES ('OR', 'WA') TABLESPACE tbs_1,
```

```
     SUBPARTITION southwest VALUES ('AZ', 'UT', 'NM') TABLESPACE tbs_2,
     SUBPARTITION northeast VALUES ('NY', 'VM', 'NJ') TABLESPACE tbs_3,
     SUBPARTITION southeast VALUES ('FL', 'GA') TABLESPACE tbs_4,
     SUBPARTITION midwest VALUES ('SD', 'WI') TABLESPACE tbs_5,
     SUBPARTITION south VALUES ('AL', 'AK') TABLESPACE tbs_6,
     SUBPARTITION others VALUES (DEFAULT ) TABLESPACE tbs_7
    )
(PARTITION q1_1999 VALUES LESS THAN ( TO_DATE('01-APR-1999','DD-MON-YYYY')),
 PARTITION q2_1999 VALUES LESS THAN ( TO_DATE('01-JUL-1999','DD-MON-YYYY')),
 PARTITION q3_1999 VALUES LESS THAN ( TO_DATE('01-OCT-1999','DD-MON-YYYY')),
 PARTITION q4_1999 VALUES LESS THAN ( TO_DATE('1-JAN-2000','DD-MON-YYYY'))
);
```

If you specified the tablespaces at the partition level (for example, `tbs_1` for partition `q1_1999`, `tbs_2` for partition `q1_1999`, `tbs_3` for partition `q3_1999`, and `tbs_4` for partition `q4_1999`) and not in the subpartition template, then the table would be horizontally striped. All subpartitions would be in the tablespace of the owning partition.

Using Multicolumn Partitioning Keys

For range- and hash-partitioned tables, you can specify up to 16 partitioning key columns. Multicolumn partitioning should be used when the partitioning key is composed of several columns and subsequent columns define a higher granularity than the preceding ones. The most common scenario is a decomposed `DATE` or `TIMESTAMP` key, consisting of separated columns, for year, month, and day.

In evaluating multicolumn partitioning keys, the database uses the second value only if the first value cannot uniquely identify a single target partition, and uses the third value only if the first and second do not determine the correct partition, and so forth. A value cannot determine the correct partition only when a partition bound exactly matches that value and the same bound is defined for the next partition. The n^{th} column will therefore be investigated only when all previous (n-1) values of the multicolumn key exactly match the (n-1) bounds of a partition. A second column, for example, will be evaluated only if the first column exactly matches the partition boundary value. If all column values exactly match all of the bound values for a partition, the database will determine that the row does not fit in this partition and will consider the next partition for a match.

In the case of nondeterministic boundary definitions (successive partitions with identical values for at least one column), the partition boundary value becomes an inclusive value, representing a "less than or equal to" boundary. This is in contrast

to deterministic boundaries, where the values are always regarded as "less than" boundaries.

The following example illustrates the column evaluation for a multicolumn range-partitioned table, storing the actual DATE information in three separate columns: year, month, and date. The partitioning granularity is a calendar quarter. The partitioned table being evaluated is created as follows:

```
CREATE TABLE sales_demo (
   year          NUMBER,
   month         NUMBER,
   day           NUMBER,
   amount_sold   NUMBER)
PARTITION BY RANGE (year,month)
  (PARTITION before2001 VALUES LESS THAN (2001,1),
   PARTITION q1_2001    VALUES LESS THAN (2001,4),
   PARTITION q2_2001    VALUES LESS THAN (2001,7),
   PARTITION q3_2001    VALUES LESS THAN (2001,10),
   PARTITION q4_2001    VALUES LESS THAN (2002,1),
   PARTITION future     VALUES LESS THAN (MAXVALUE,0));

REM  12-DEC-2000
INSERT INTO sales_demo VALUES(2000,12,12, 1000);
REM  17-MAR-2001
INSERT INTO sales_demo VALUES(2001,3,17, 2000);
REM  1-NOV-2001
INSERT INTO sales_demo VALUES(2001,11,1, 5000);
REM  1-JAN-2002
INSERT INTO sales_demo VALUES(2002,1,1, 4000);
```

The year value for 12-DEC-2000 satisfied the first partition, before2001, so no further evaluation is needed:

```
SELECT * FROM sales_demo PARTITION(before2001);

      YEAR      MONTH        DAY AMOUNT_SOLD
---------- ---------- ---------- -----------
      2000         12         12        1000
```

The information for 17-MAR-2001 is stored in partition q1_2001. The first partitioning key column, year, does not by itself determine the correct partition, so the second partition key column, month, must be evaluated.

```
SELECT * FROM sales_demo PARTITION(q1_2001);

      YEAR      MONTH        DAY AMOUNT_SOLD
```

```
---------- ---------- ---------- -----------
      2001          3         17        2000
```

Following the same determination rule as for the previous record, the second column, `month`, determines partition `q4_2001` as correct partition for 1-NOV-2001:

```
SELECT * FROM sales_demo PARTITION(q4_2001);

      YEAR      MONTH        DAY AMOUNT_SOLD
---------- ---------- ---------- -----------
      2001         11          1        5000
```

The partition for 01-JAN-2002 is determined by evaluating only the `year` column, which indicates the `future` partition:

```
SELECT * FROM sales_demo PARTITION(future);

      YEAR      MONTH        DAY AMOUNT_SOLD
---------- ---------- ---------- -----------
      2002          1          1        4000
```

If the database encounters MAXVALUE in one of the partition key columns, all other values of subsequent columns become irrelevant. That is, a definition of partition `future` in the preceding example, having a bound of (MAXVALUE,0) is equivalent to a bound of (MAXVALUE,100) or a bound of (MAXVALUE,MAXVALUE).

The following example illustrates the use of a multicolumn partitioned approach for table `supplier_parts`, storing the information about which suppliers deliver which parts. To distribute the data in equal-sized partitions, it is not sufficient to partition the table based on the `supplier_id`, because some suppliers might provide hundreds of thousands of parts, while others provide only a few specialty parts. Instead, you partition the table on (`supplier_id`, `partnum`) to manually enforce equal-sized partitions.

```
CREATE TABLE supplier_parts (
   supplier_id      NUMBER,
   partnum          NUMBER,
   price            NUMBER)
PARTITION BY RANGE (supplier_id, partnum)
  (PARTITION p1 VALUES LESS THAN  (10,100),
   PARTITION p2 VALUES LESS THAN (10,200),
   PARTITION p3 VALUES LESS THAN (MAXVALUE,MAXVALUE));
```

The following three records are inserted into the table:

```
INSERT INTO supplier_parts VALUES (5,5, 1000);
```

```
INSERT INTO supplier_parts VALUES (5,150, 1000);
INSERT INTO supplier_parts VALUES (10,100, 1000);
```

The first two records are inserted into partition p1, uniquely identified by supplier_id. However, the third record is inserted into partition p2; it matches all range boundary values of partition p1 exactly and the database therefore considers the following partition for a match. The value of partnum satisfies the criteria < 200, so it is inserted into partition p2.

```
SELECT * FROM supplier_parts PARTITION (p1);

SUPPLIER_ID    PARTNUM      PRICE
----------- ---------- ----------
          5          5       1000
          5        150       1000

SELECT * FROM supplier_parts PARTITION (p2);

SUPPLIER_ID    PARTNUM      PRICE
----------- ---------- ----------
          10       100       1000
```

Every row with supplier_id < 10 will be stored in partition p1, regardless of the partnum value. The column partnum will be evaluated only if supplier_id =10, and the corresponding rows will be inserted into partition p1, p2, or even into p3 when partnum >=200. To achieve equal-sized partitions for ranges of supplier_parts, you could choose a composite range-hash partitioned table, range partitioned by supplier_id, hash subpartitioned by partnum.

Defining the partition boundaries for multicolumn partitioned tables must obey some rules. For example, consider a table that is range partitioned on three columns a, b, and c. The individual partitions have range values represented as follows:

```
P0(a0, b0, c0)
P1(a1, b1, c1)
P2(a2, b2, c2)
…
Pn(an, bn, cn)
```

The range values you provide for each partition must follow these rules:

- a0 must be less than or equal to a1, and a1 must be less than or equal to a2, and so on.

- If a0=a1, then b0 must be less than or equal to b1. If a0 < a1, then b0 and b1 can have any values. If b0=b1, then c0 must be less than or equal to c1. If b0<b1, then c0 and c1 can have any values, and so on.
- If a1=a2, then b1 must be less than or equal to b2. If a1<a2, then b1 and b2 can have any values. If b1=b2, then c1 must be less than or equal to c2. If b1<b2, then c0 and c1 can have any values, and so on.

Using Table Compression with Partitioned Tables

For heap-organized partitioned tables, you can compress some or all partitions using table compression. The compression attribute can be declared for a tablespace, a table, or a partition of a table. Whenever the compress attribute is not specified, it is inherited like any other storage attribute.

The following example creates a list-partitioned table with one compressed partition costs_old. The compression attribute for the table and all other partitions is inherited from the tablespace level.

```
CREATE TABLE costs_demo (
   prod_id     NUMBER(6),    time_id     DATE,
   unit_cost   NUMBER(10,2), unit_price  NUMBER(10,2))
PARTITION BY RANGE (time_id)
   (PARTITION costs_old
       VALUES LESS THAN (TO_DATE('01-JAN-2003', 'DD-MON-YYYY')) COMPRESS,
    PARTITION costs_q1_2003
       VALUES LESS THAN (TO_DATE('01-APR-2003', 'DD-MON-YYYY')),
    PARTITION costs_q2_2003
       VALUES LESS THAN (TO_DATE('01-JUN-2003', 'DD-MON-YYYY')),
    PARTITION costs_recent VALUES LESS THAN (MAXVALUE));
```

Using Key Compression with Partitioned Indexes

You can compress some or all partitions of a B-tree index using key compression. Key compression is applicable only to B-tree indexes. Bitmap indexes are stored in a compressed manner by default. Index using key compression eliminates repeated occurrences of key column prefix values, thus saving space and I/O.

The following example crates a local partitioned index with all partitions except the most recent one compressed:

```
CREATE INDEX i_cost1 ON costs_demo (prod_id) COMPRESS LOCAL
   (PARTITION costs_old, PARTITION costs_q1_2003,
    PARTITION costs_q2_2003, PARTITION costs_recent NOCOMPRESS);
```

You cannot specify COMPRESS (or NOCOMPRESS) explicitly for an index subpartition. All index subpartitions of a given partition inherit the key compression setting from the parent partition.

To modify the key compression attribute for all subpartitions of a given partition, you must first issue an ALTER INDEX ... MODIFY PARTITION statement and then rebuild all subpartitions. The MODIFY PARTITION clause will mark all index subpartitions as UNUSABLE.

Creating Partitioned Index-Organized Tables

For index-organized tables, you can use the range, list, or hash partitioning method. The semantics for creating partitioned index-organized tables is similar to that for regular tables with these differences:

- When you create the table you specify the ORGANIZATION INDEX clause, and INCLUDING and OVERFLOW clauses as necessary.
- The PARTITION or PARTITIONS clauses can have OVERFLOW subclauses that allow you to specify attributes of the overflow segments at the partition level.

Specifying an OVERFLOW clause results in the overflow data segments themselves being equipartitioned with the primary key index segments. Thus, for partitioned index-organized tables with overflow, each partition has an index segment and an overflow data segment.

For index-organized tables, the set of partitioning columns must be a subset of the primary key columns. Since rows of an index-organized table are stored in the primary key index for the table, the partitioning criterion has an effect on the availability. By choosing the partition key to be a subset of the primary key, an insert operation only needs to verify uniqueness of the primary key in a single partition, thereby maintaining partition independence.

Support for secondary indexes on index-organized tables is similar to the support for regular tables. Because of the logical nature of the secondary indexes, global indexes on index-organized tables remain usable for certain operations where they would be marked UNUSABLE for regular tables.

See Also:

- "Managing Index-Organized Tables" on page 14-40
- "Maintaining Partitioned Tables" on page 16-28
- *Oracle Database Concepts* for more information about index-organized tables

Creating Range-Partitioned Index-Organized Tables

You can partition index-organized tables, and their secondary indexes, by the range method. In the following example, a range-partitioned index-organized table `sales` is created. The `INCLUDING` clause specifies all columns after `week_no` are stored in an overflow segment. There is one overflow segment for each partition, all stored in the same tablespace (`overflow_here`). Optionally, `OVERFLOW TABLESPACE` could be specified at the individual partition level, in which case some or all of the overflow segments could have separate `TABLESPACE` attributes.

```
CREATE TABLE sales(acct_no NUMBER(5),
                   acct_name CHAR(30),
                   amount_of_sale NUMBER(6),
                   week_no INTEGER,
                   sale_details VARCHAR2(1000),
             PRIMARY KEY (acct_no, acct_name, week_no))
     ORGANIZATION INDEX
             INCLUDING week_no
             OVERFLOW TABLESPACE overflow_here
     PARTITION BY RANGE (week_no)
            (PARTITION VALUES LESS THAN (5)
                   TABLESPACE ts1,
             PARTITION VALUES LESS THAN (9)
                   TABLESPACE ts2 OVERFLOW TABLESPACE overflow_ts2,
             ...
             PARTITION VALUES LESS THAN (MAXVALUE)
                   TABLESPACE ts13);
```

Creating List-Partitioned Index-Organized Tables

Another option for partitioning index-organized tables is to use the list method. In the following example the index-organized table, `sales`, is partitioned by the list method. This example uses the `example` tablespace, which is part of the sample schemas in your seed database. Normally you would specify different tablespace storage for different partitions.

```
CREATE TABLE sales(acct_no NUMBER(5),
                   acct_name CHAR(30),
                   amount_of_sale NUMBER(6),
                   week_no INTEGER,
                   sale_details VARCHAR2(1000),
             PRIMARY KEY (acct_no, acct_name, week_no))
     ORGANIZATION INDEX
             INCLUDING week_no
             OVERFLOW TABLESPACE example
     PARTITION BY LIST (week_no)
```

```
            (PARTITION VALUES (1, 2, 3, 4)
                  TABLESPACE example,
             PARTITION VALUES (5, 6, 7, 8)
                  TABLESPACE example OVERFLOW TABLESPACE example,
             PARTITION VALUES (DEFAULT)
                  TABLESPACE example);
```

Creating Hash-Partitioned Index-Organized Tables

The other option for partitioning index-organized tables is to use the hash method. In the following example the index-organized table, `sales`, is partitioned by the hash method.

```
CREATE TABLE sales(acct_no NUMBER(5),
                   acct_name CHAR(30),
                   amount_of_sale NUMBER(6),
                   week_no INTEGER,
                   sale_details VARCHAR2(1000),
              PRIMARY KEY (acct_no, acct_name, week_no))
     ORGANIZATION INDEX
             INCLUDING week_no
     OVERFLOW
          PARTITION BY HASH (week_no)
             PARTITIONS 16
             STORE IN (ts1, ts2, ts3, ts4)
             OVERFLOW STORE IN (ts3, ts6, ts9);
```

Note: A well-designed hash function is intended to distribute rows in a well-balanced fashion among the partitions. Therefore, updating the primary key column(s) of a row is very likely to move that row to a different partition. Oracle recommends that you explicitly specify the `ROW MOVEMENT ENABLE` clause when creating a hash-partitioned index-organized table with a changeable partitioning key. The default is that `ROW MOVEMENT ENABLE` is disabled.

Partitioning Restrictions for Multiple Block Sizes

Use caution when creating partitioned objects in a database with tablespaces of multiple block size. The storage of partitioned objects in such tablespaces is subject to some restrictions. Specifically, all partitions of the following entities must reside in tablespaces of the same block size:

- Conventional tables
- Indexes
- Primary key index segments of index-organized tables
- Overflow segments of index-organized tables
- LOB columns stored out of line

Therefore:

- For each conventional table, all partitions of that table must be stored in tablespaces with the same block size.
- For each index-organized table, all primary key index partitions must reside in tablespaces of the same block size, and all overflow partitions of that table must reside in tablespaces of the same block size. However, index partitions and overflow partitions can reside in tablespaces of different block size.
- For each index (global or local), each partition of that index must reside in tablespaces of the same block size. However, partitions of different indexes defined on the same object can reside in tablespaces of different block sizes.
- For each LOB column, each partition of that column must be stored in tablespaces of equal block sizes. However, different LOB columns can be stored in tablespaces of different block sizes.

When you create or alter a partitioned table or index, all tablespaces you *explicitly specify* for the partitions and subpartitions of each entity must be of the same block size. If you *do not explicitly specify* tablespace storage for an entity, the tablespaces the database uses by default must be of the same block size. Therefore you must be aware of the default tablespaces at each level of the partitioned object.

Maintaining Partitioned Tables

This section describes how to perform partition and subpartition maintenance operations for both tables and indexes.

Table 16–1 lists maintenance operations that can be performed on table partitions (or subpartitions) and, for each type of partitioning, lists the specific clause of the ALTER TABLE statement that is used to perform that maintenance operation.

Table 16–1 ALTER TABLE Maintenance Operations for Table Partitions

Maintenance Operation	Range	Hash	List	Composite: Range/Hash	Composite: Range/List
Adding Partitions	ADD PARTITION	ADD PARTITION	ADD PARTITION	ADD PARTITION MODIFY PARTITION ... ADD SUBPARTITION	ADD PARTITION MODIFY PARTITION ... ADD SUBPARTITION
Coalescing Partitions	n/a	COALESCE PARTITION	n/a	MODIFY PARTITION ... COALESCE SUBPARTITION	n/a
Dropping Partitions	DROP PARTITION	n/a	DROP PARTITION	DROP PARTITION	DROP PARTITION DROP SUBPARTITION
Exchanging Partitions	EXCHANGE PARTITION	EXCHANGE PARTITION	EXCHANGE PARTITION	EXCHANGE PARTITION EXCHANGE SUBPARTITION	EXCHANGE PARTITION EXCHANGE SUBPARTITION
Merging Partitions	MERGE PARTITIONS	n/a	MERGE PARTITIONS	MERGE PARTITIONS	MERGE PARTITIONS MERGE SUBPARTITIONS
Modifying Default Attributes	MODIFY DEFAULT ATTRIBUTES	MODIFY DEFAULT ATTRIBUTES	MODIFY DEFAULT ATTRIBUTES	MODIFY DEFAULT ATTRIBUTES MODIFY DEFAULT ATTRIBUTES FOR PARTITION	MODIFY DEFAULT ATTRIBUTES MODIFY DEFAULT ATTRIBUTES FOR PARTITION
Modifying Real Attributes of Partitions	MODIFY PARTITION	MODIFY PARTITION	MODIFY PARTITION	MODIFY PARTITION MODIFY SUBPARTITION	MODIFY PARTITION MODIFY SUBPARTITION
Modifying List Partitions: Adding Values	n/a	n/a	MODIFY PARTITION... ADD VALUES	n/a	MODIFY SUBPARTITION ... ADD VALUES
Modifying List Partitions: Dropping Values	n/a	n/a	MODIFY PARTITION... DROP VALUES	n/a	MODIFY SUBPARTITION ... DROP VALUES
Modifying a Subpartition Template	n/a	n/a	n/a	SET SUBPARTITION TEMPLATE	SET SUBPARTITION TEMPLATE
Moving Partitions	MOVE PARTITION	MOVE PARTITION	MOVE PARTITION	MOVE SUBPARTITION	MOVE SUBPARTITION

Table 16–1 (Cont.) ALTER TABLE Maintenance Operations for Table Partitions

Maintenance Operation	Range	Hash	List	Composite: Range/Hash	Composite: Range/List
Renaming Partitions	RENAME PARTITION	RENAME PARTITION	RENAME PARTITION	RENAME PARTITION RENAME SUBPARTITION	RENAME PARTITION RENAME SUBPARTITION
Splitting Partitions	SPLIT PARTITION	n/a	SPLIT PARTITION	SPLIT PARTITION	SPLIT PARTITION SPLIT SUBPARTITION
Truncating Partitions	TRUNCATE PARTITION	TRUNCATE PARTITION	TRUNCATE PARTITION	TRUNCATE PARTITION TRUNCATE SUBPARTITION	TRUNCATE PARTITION TRUNCATE SUBPARTITION

Note: The first time you use table compression to introduce a compressed partition into a partitioned table that has bitmap indexes and that currently contains only uncompressed partitions, you must do the following:

- Either drop all existing bitmap indexes and bitmap index partitions, or mark them UNUSABLE.
- Set the table compression attribute.
- Rebuild the indexes.

These actions are independent of whether any partitions contain data and of the operation that introduces the compressed partition.

This does not apply to partitioned tables with B-tree indexes or to partitioned index-organized tables.

For more information, see the *Oracle Data Warehousing Guide*.

Table 16–2 lists maintenance operations that can be performed on index partitions, and indicates on which type of index (global or local) they can be performed. The ALTER INDEX clause used for the maintenance operation is shown.

Global indexes do not reflect the structure of the underlying table. If partitioned, they can be partitioned by range or hash. Partitioned global indexes share some, but not all, of the partition maintenance operations that can be performed on partitioned tables.

Because local indexes reflect the underlying structure of the table, partitioning is maintained automatically when table partitions and subpartitions are affected by maintenance activity. Therefore, partition maintenance on local indexes is less necessary and there are fewer options.

Table 16–2 ALTER INDEX Maintenance Operations for Index Partitions

Maintenance Operation	Type of Index	Type of Index Partitioning		
		Range	**Hash and List**	**Composite**
Adding Index Partitions	Global	-	`ADD PARTITION` (hash only)	-
	Local	n/a	n/a	n/a
Dropping Index Partitions	Global	`DROP PARTITION`	-	-
	Local	n/a	n/a	n/a
Modifying Default Attributes of Index Partitions	Global	`MODIFY DEFAULT ATTRIBUTES`	-	-
	Local	`MODIFY DEFAULT ATTRIBUTES`	`MODIFY DEFAULT ATTRIBUTES`	`MODIFY DEFAULT ATTRIBUTES` `MODIFY DEFAULT ATTRIBUTES FOR PARTITION`
Modifying Real Attributes of Index Partitions	Global	`MODIFY PARTITION`	-	-
	Local	`MODIFY PARTITION`	`MODIFY PARTITION`	`MODIFY PARTITION` `MODIFY SUBPARTITION`
Rebuilding Index Partitions	Global	`REBUILD PARTITION`	-	-
	Local	`REBUILD PARTITION`	`REBUILD PARTITION`	`REBUILD SUBPARTITION`
Renaming Index Partitions	Global	`RENAME PARTITION`	-	-
	Local	`RENAME PARTITION`	`RENAME PARTITION`	`RENAME PARTITION` `RENAME SUBPARTITION`
Splitting Index Partitions	Global	`SPLIT PARTITION`	-	-
	Local	n/a	n/a	n/a

Note: The following sections discuss maintenance operations on partitioned tables. Where the usability of indexes or index partitions affected by the maintenance operation is discussed, consider the following:

- Only indexes and index partitions that are *not* empty are candidates for being marked UNUSABLE. If they are empty, the USABLE / UNUSABLE status is left unchanged.
- Only indexes or index partitions with USABLE status are updated by subsequent DML.

Updating Indexes Automatically

Before discussing the individual maintenance operations for partitioned tables and indexes, it is important to discuss the effects of the UPDATE INDEXES clause that can be specified in the ALTER TABLE statement.

By default, many table maintenance operations on partitioned tables invalidate (mark UNUSABLE) the corresponding indexes or index partitions. You must then rebuild the entire index or, in the case of a global index, each of its partitions. The database lets you override this default behavior if you specify UPDATE INDEXES in your ALTER TABLE statement for the maintenance operation. Specifying this clause tells the database to update the index at the time it executes the maintenance operation DDL statement. This provides the following benefits:

- The index is updated in conjunction with the base table operation. You are not required to later and independently rebuild the index.
- The index is more highly available, because it does not get marked UNUSABLE. The index remains available even while the partition DDL is executing and it can be used to access unaffected partitions in the table.
- You need not look up the names of all invalid indexes to rebuild them.

Optional clauses for local indexes let you specify physical and storage characteristics for updated local indexes and their partitions.

- You can specify physical attributes, tablespace storage, and logging for each partition of each local index. Alternatively, you can specify only the PARTITION keyword and let the database update the as follows
  - For operations on a single table partition (such as MOVE PARTITION and SPLIT PARTITION), the corresponding index partition inherits the attributes of the affected index partition. The database does not generate

names for new index partitions, so any new index partitions resulting from this operation inherit their names from the corresponding new table partition.

  - For MERGE PARTITION operations, the resulting local index partition inherits its name from the resulting table partition and inherits its attributes from the local index.

- For a composite-partitioned index, you can specify tablespace storage for each subpartition.

  See Also: the *update_all_indexes_clause* of ALTER TABLE for the syntax for updating indexes

The following operations support the UPDATE INDEXES clause:

- ADD PARTITION | SUBPARTITION
- COALESCE PARTITION | SUBPARTITION
- DROP PARTITION | SUBPARTITION
- EXCHANGE PARTITION | SUBPARTITION
- MERGE PARTITION | SUBPARTITION
- MOVE PARTITION | SUBPARTITION
- SPLIT PARTITION | SUBPARTITION
- TRUNCATE PARTITION | SUBPARTITION

SKIP_UNUSABLE_INDEXES Initialization Parameter

The SKIP_UNUSABLE_INDEXES, which in earlier releases was a session parameter, is now an initialization parameter with a default value of TRUE. This setting disables error reporting of indexes and index partitions marked UNUSABLE. If you do not want the database to choose an alternative execution plan to avoid the unusable elements, you should set this parameter to FALSE.

Considerations when Updating Indexes Automatically

The following performance implications are worth noting when you specify UPDATE INDEXES:

- The partition DDL statement takes longer to execute, because indexes that were previously marked UNUSABLE are updated. However, you must compare this increase with the time it takes to execute DDL without updating indexes, and

then rebuild all indexes. A rule of thumb is that it is faster to update indexes if the size of the partition is less that 5% of the size of the table.

- The DROP, TRUNCATE, and EXCHANGE operations are no longer fast operations. Again, you must compare the time it takes to do the DDL and then rebuild all indexes.
- When you update a table with a global index:
  - The index is updated in place. The updates to the index are logged, and redo and undo records are generated. In contrast, if you rebuild an entire global index, you can do so in NOLOGGING mode.
  - Rebuilding the entire index manually creates a more efficient index, because it is more compact with space better utilized.

Adding Partitions

This section describes how to add new partitions to a partitioned table and explains why partitions cannot be specifically added to most partitioned indexes.

Adding a Partition to a Range-Partitioned Table

Use the ALTER TABLE ... ADD PARTITION statement to add a new partition to the "high" end (the point after the last existing partition). To add a partition at the beginning or in the middle of a table, use the SPLIT PARTITION clause.

For example, consider the table, sales, which contains data for the current month in addition to the previous 12 months. On January 1, 1999, you add a partition for January, which is stored in tablespace tsx.

```
ALTER TABLE sales
      ADD PARTITION jan96 VALUES LESS THAN ( '01-FEB-1999' )
      TABLESPACE tsx;
```

Local and global indexes associated with the range-partitioned table remain usable.

Adding a Partition to a Hash-Partitioned Table

When you add a partition to a hash-partitioned table, the database populates the new partition with rows rehashed from an existing partition (selected by the database) as determined by the hash function.

The following statements show two ways of adding a hash partition to table scubagear. Choosing the first statement adds a new hash partition whose partition name is system generated, and which is placed in the table default

tablespace. The second statement also adds a new hash partition, but that partition is explicitly named p_named and is created in tablespace gear5.

```
ALTER TABLE scubagear ADD PARTITION;

ALTER TABLE scubagear
      ADD PARTITION p_named TABLESPACE gear5;
```

Indexes may be marked UNUSABLE as explained in the following table:

Table Type	Index Behavior
Regular (Heap)	Unless you specify UPDATE INDEXES as part of the ALTER TABLE statement: ■ The local indexes for the new partition, and for the existing partition from which rows were redistributed, are marked UNUSABLE and must be rebuilt. ■ All global indexes, or all partitions of partitioned global indexes, are marked UNUSABLE and must be rebuilt.
Index-organized	■ For local indexes, the behavior is the same as for heap tables. ■ All global indexes remain usable.

Adding a Partition to a List-Partitioned Table

The following statement illustrates adding a new partition to a list-partitioned table. In this example physical attributes and NOLOGGING are specified for the partition being added.

```
ALTER TABLE q1_sales_by_region
   ADD PARTITION q1_nonmainland VALUES ('HI', 'PR')
      STORAGE (INITIAL 20K NEXT 20K) TABLESPACE tbs_3
      NOLOGGING;
```

Any value in the set of literal values that describe the partition being added must not exist in any of the other partitions of the table.

You cannot add a partition to a list-partitioned table that has a default partition, but you can split the default partition. By doing so, you effectively create a new partition defined by the values that you specify, and a second partition that remains the default partition.

Local and global indexes associated with the list-partitioned table remain usable.

Adding Partitions to a Range-Hash Composite-Partitioned Table

Partitions can be added at both the range partition level and the hash subpartition level.

Adding a Partition to a Range-Hash Composite-Partitioned Table Adding a new range partition to a range-hash partitioned table is as described previously in "Adding a Partition to a Range-Partitioned Table". However, you can specify a SUBPARTITIONS clause that lets you add a specified number of subpartitions, or a SUBPARTITION clause for naming specific subpartitions. If no SUBPARTITIONS or SUBPARTITION clause is specified, the partition inherits table level defaults for subpartitions.

This example adds a range partition `q1_2000` to table `sales`, which will be populated with data for the first quarter of the year 2000. There are eight subpartitions stored in tablespace `tbs5`. The subpartitions cannot be set explicitly to use table compression. Subpartitions inherit the compression attribute from the partition level and are stored in a compressed form in this example:

```
ALTER TABLE sales ADD PARTITION q1_2000
      VALUES LESS THAN (2000, 04, 01) COMPRESS
      SUBPARTITIONS 8 STORE IN tbs5;
```

Adding a Subpartition to a Range-Hash Partitioned Table You use the MODIFY PARTITION ... ADD SUBPARTITION clause of the ALTER TABLE statement to add a hash subpartition to a range-hash partitioned table. The newly added subpartition is populated with rows rehashed from other subpartitions of the same partition as determined by the hash function.

In the following example, a new hash subpartition `us_loc5`, stored in tablespace `us1`, is added to range partition `locations_us` in table `diving`.

```
ALTER TABLE diving MODIFY PARTITION locations_us
      ADD SUBPARTITION us_locs5 TABLESPACE us1;
```

Index subpartitions corresponding to the added and rehashed subpartitions must be rebuilt unless you specify UPDATE INDEXES.

Adding Partitions to a Range-List Partitioned Table

Partitions can be added at both the range partition level and the list subpartition level.

Adding a Partition to a Range-List Partitioned Table Adding a new range partition to a range-list partitioned table is as described previously in "Adding a Partition to a

Range-Partitioned Table". However, you can specify SUBPARTITION clauses for naming and providing value lists for the subpartitions. If no SUBPARTITION clauses are specified, then the partition inherits the subpartition template. If there is no subpartition template, then a single default subpartition is created.

This following statement statements adds a new partition to the quarterly_regional_sales table that is partitioned by the range-list method. Some new physical attributes are specified for this new partition while table-level defaults are inherited for those that are not specified.

```
ALTER TABLE quarterly_regional_sales
   ADD PARTITION q1_2000 VALUES LESS THAN (TO_DATE('1-APR-2000','DD-MON-YYYY'))
      STORAGE (INITIAL 20K NEXT 20K) TABLESPACE ts3 NOLOGGING
         (
          SUBPARTITION q1_2000_northwest VALUES ('OR', 'WA'),
          SUBPARTITION q1_2000_southwest VALUES ('AZ', 'UT', 'NM'),
          SUBPARTITION q1_2000_northeast VALUES ('NY', 'VM', 'NJ'),
          SUBPARTITION q1_2000_southeast VALUES ('FL', 'GA'),
          SUBPARTITION q1_2000_northcentral VALUES ('SD', 'WI'),
          SUBPARTITION q1_2000_southcentral VALUES ('OK', 'TX')
         );
```

Adding a Subpartition to a Range-List Partitioned Table You use the MODIFY PARTITION ... ADD SUBPARTITION clause of the ALTER TABLE statement to add a list subpartition to a range-list partitioned table.

The following statement adds a new subpartition to the existing set of subpartitions in range-list partitioned table quarterly_regional_sales. The new subpartition is created in tablespace ts2.

```
ALTER TABLE quarterly_regional_sales
   MODIFY PARTITION q1_1999
      ADD SUBPARTITION q1_1999_south
         VALUES ('AR','MS','AL') tablespace ts2;
```

Adding Index Partitions

You cannot explicitly add a partition to a local index. Instead, a new partition is added to a local index only when you add a partition to the underlying table. Specifically, when there is a local index defined on a table and you issue the ALTER TABLE statement to add a partition, a matching partition is also added to the local index. The database assigns names and default physical storage attributes to the new index partitions, but you can rename or alter them after the ADD PARTITION operation is complete.

You can effectively specify a new tablespace for an index partition in an ADD PARTITION operation by first modifying the default attributes for the index. For example, assume that a local index, q1_sales_by_region_locix, was created for list partitioned table q1_sales_by_region. If before adding the new partition q1_nonmainland, as shown in "Adding a Partition to a List-Partitioned Table" on page 16-35, you had issued the following statement, then the corresponding index partition would be created in tablespace tbs_4.

```
ALTER INDEX q1_sales_by_region_locix
   MODIFY DEFAULT ATTRIBUTES TABLESPACE tbs_4;
```

Otherwise, it would be necessary for you to use the following statement to move the index partition to tbs_4 after adding it:

```
ALTER INDEX q1_sales_by_region_locix
   REBUILD PARTITION q1_nonmainland TABLESPACE tbs_4;
```

You can add a partition to a hash-partitioned global index using the ADD PARTITION syntax of ALTER INDEX. The database adds hash partitions and populates them with index entries rehashed from an existing hash partition of the index, as determined by the hash function. The following statement adds a partition to the index hgidx shown in "Creating a Hash-Partitioned Global Index" on page 16-13:

```
ALTER INDEX hgidx ADD PARTITION p5;
```

You cannot add a partition to a range-partitioned global index, because the highest partition always has a partition bound of MAXVALUE. If you want to add a new highest partition, use the ALTER INDEX ... SPLIT PARTITION statement.

Coalescing Partitions

Coalescing partitions is a way of reducing the number of partitions in a hash-partitioned table or index, or the number of subpartitions in a range-hash partitioned table. When a hash partition is coalesced, its contents are redistributed into one or more remaining partitions determined by the hash function. The specific partition that is coalesced is selected by the database, and is dropped after its contents have been redistributed.

Index partitions may be marked UNUSABLE as explained in the following table:

Table Type	Index Behavior
Regular (Heap)	Unless you specify UPDATE INDEXES as part of the ALTER TABLE statement: ■ Any local index partition corresponding to the selected partition is also dropped. Local index partitions corresponding to the one or more absorbing partitions are marked UNUSABLE and must be rebuilt. ■ All global indexes, or all partitions of partitioned global indexes, are marked UNUSABLE and must be rebuilt.
Index-organized	■ Some local indexes are marked UNUSABLE as noted for heap indexes. ■ All global indexes remain usable.

Coalescing a Partition in a Hash-Partitioned Table

The ALTER TABLE ... COALESCE PARTITION statement is used to coalesce a partition in a hash-partitioned table. The following statement reduces by one the number of partitions in a table by coalescing a partition.

```
ALTER TABLE ouu1
     COALESCE PARTITION;
```

Coalescing a Subpartition in a Range-Hash Partitioned Table

The following statement distributes the contents of a subpartition of partition us_locations into one or more remaining subpartitions (determined by the hash function) of the same partition. Basically, this operation is the inverse of the MODIFY PARTITION ... ADD SUBPARTITION clause discussed in "Adding a Subpartition to a Range-Hash Partitioned Table" on page 16-36.

```
ALTER TABLE diving MODIFY PARTITION us_locations
     COALESCE SUBPARTITION;
```

Coalescing Hash-partitioned Global Indexes

You can instruct the database to reduce by one the number of index partitions in a hash-partitioned global index using the COALESCE PARTITION clause of ALTER INDEX. The database selects the partition to coalesce based on the requirements of the hash partition. The following statement reduces by one the number of partitions in the hgidx index, created in "Creating a Hash-Partitioned Global Index" on page 16-13:

```
ALTER INDEX hgidx COALESCE PARTITION;
```

Dropping Partitions

You can drop partitions from range, list, or composite range-list partitioned tables. For hash-partitioned tables, or hash subpartitions of range-hash partitioned tables, you must perform a coalesce operation instead.

Dropping a Table Partition

Use one of the following statements to drop a table partition or subpartition:

- ALTER TABLE ... DROP PARTITION to drop a table partition
- ALTER TABLE ... DROP SUBPARTITION to drop a subpartition of a range-list partitioned table

If you want to preserve the data in the partition, use the MERGE PARTITION statement instead of the DROP PARTITION statement.

If local indexes are defined for the table, this statement also drops the matching partition or subpartitions from the local index. All global indexes, or all partitions of partitioned global indexes, are marked UNUSABLE unless either of the following are true:

- You specify UPDATE INDEXES (cannot be specified for index-organized tables)
- The partition being dropped or its subpartitions are empty

> **Note:** You cannot drop the only partition in a table. Instead, you must drop the table.

The following sections contain some scenarios for dropping table partitions.

Dropping a Partition from a Table that Contains Data and Global Indexes If the partition contains data and one or more global indexes are defined on the table, use one of the following methods to drop the table partition.

Method 1

Leave the global indexes in place during the ALTER TABLE ... DROP PARTITION statement. Afterward, you must rebuild any global indexes (whether partitioned or not) because the index (or index partitions) will have been marked UNUSABLE. The following statements provide an example of dropping partition dec98 from the sales table, then rebuilding its global nonpartitioned index.

```
ALTER TABLE sales DROP PARTITION dec98;
```

```
ALTER INDEX sales_area_ix REBUILD;
```

If index sales_area_ix were a range-partitioned global index, then all partitions of the index would require rebuilding. Further, it is not possible to rebuild all partitions of an index in one statement. You must write a separate REBUILD statement for each partition in the index. The following statements rebuild the index partitions jan99_ix, feb99_ix, mar99_ix, ..., dec99_ix.

```
ALTER INDEX sales_area_ix REBUILD PARTITION jan99_ix;
ALTER INDEX sales_area_ix REBUILD PARTITION feb99_ix;
ALTER INDEX sales_area_ix REBUILD PARTITION mar99_ix;
...
ALTER INDEX sales_area_ix REBUILD PARTITION dec99_ix;
```

This method is most appropriate for large tables where the partition being dropped contains a significant percentage of the total data in the table.

Method 2

Issue the DELETE statement to delete all rows from the partition before you issue the ALTER TABLE ... DROP PARTITION statement. The DELETE statement updates the global indexes, and also fires triggers and generates redo and undo logs.

For example, to drop the first partition, which has a partition bound of 10000, issue the following statements:

```
DELETE FROM sales WHERE TRANSID < 10000;
ALTER TABLE sales DROP PARTITION dec98;
```

This method is most appropriate for small tables, or for large tables when the partition being dropped contains a small percentage of the total data in the table.

Method 3

Specify UPDATE INDEXES in the ALTER TABLE statement. Doing so causes the global index to be updated at the time the partition is dropped.

```
ALTER TABLE sales DROP PARTITION dec98
     UPDATE INDEXES;
```

Dropping a Partition Containing Data and Referential Integrity Constraints If a partition contains data and the table has referential integrity constraints, choose either of the following methods to drop the table partition. This table has a local index only, so it is not necessary to rebuild any indexes.

Method 1

Disable the integrity constraints, issue the ALTER TABLE ... DROP PARTITION statement, then enable the integrity constraints:

```
ALTER TABLE sales
   DISABLE CONSTRAINT dname_sales1;
ALTER TABLE sales DROP PARTITTION dec98;
ALTER TABLE sales
   ENABLE CONSTRAINT dname_sales1;
```

This method is most appropriate for large tables where the partition being dropped contains a significant percentage of the total data in the table.

Method 2

Issue the DELETE statement to delete all rows from the partition before you issue the ALTER TABLE ... DROP PARTITION statement. The DELETE statement enforces referential integrity constraints, and also fires triggers and generates redo and undo log.

```
DELETE FROM sales WHERE TRANSID < 10000;
ALTER TABLE sales DROP PARTITION dec94;
```

This method is most appropriate for small tables or for large tables when the partition being dropped contains a small percentage of the total data in the table.

Dropping Index Partitions

You cannot explicitly drop a partition of a local index. Instead, local index partitions are dropped only when you drop a partition from the underlying table.

If a global index partition is empty, you can explicitly drop it by issuing the ALTER INDEX ... DROP PARTITION statement. But, if a global index partition contains data, dropping the partition causes the next highest partition to be marked UNUSABLE. For example, you would like to drop the index partition P1, and P2 is the next highest partition. You must issue the following statements:

```
ALTER INDEX npr DROP PARTITION P1;
ALTER INDEX npr REBUILD PARTITION P2;
```

Note: You cannot drop the highest partition in a global index.

Exchanging Partitions

You can convert a partition (or subpartition) into a nonpartitioned table, and a nonpartitioned table into a partition (or subpartition) of a partitioned table by exchanging their data segments. You can also convert a hash-partitioned table into a partition of a range-hash partitioned table, or convert the partition of the range-hash partitioned table into a hash-partitioned table. Similarly, you can convert a list-partitioned table into a partition of a range-list partitioned table, or convert the partition of the range-list partitioned table into a list-partitioned table

Exchanging table partitions is most useful when you have an application using nonpartitioned tables that you want to convert to partitions of a partitioned table. For example, in data warehousing environments exchanging partitions facilitates high-speed data loading of new, incremental data into an already existing partitioned table. Generically, OLTP as well as data warehousing environments benefit from exchanging old data partitions out of a partitioned table. The data is purged from the partitioned table without actually being deleted and can be archived separately afterwards.

When you exchange partitions, logging attributes are preserved. You can optionally specify if local indexes are also to be exchanged (`INCLUDING INDEXES` clause), and if rows are to be validated for proper mapping (`WITH VALIDATION` clause).

Note: When you specify `WITHOUT VALIDATION` for the exchange partition operation, this is normally a fast operation because it involves only data dictionary updates. However, if the table or partitioned table involved in the exchange operation has a primary key or unique constraint enabled, then the exchange operation will be performed as if `WITH VALIDATION` were specified in order to maintain the integrity of the constraints.

To avoid the overhead of this validation activity, issue the following statement for each constraint before doing the exchange partition operation:

```
ALTER TABLE table_name
     DISABLE CONSTRAINT constraint_name KEEP INDEX
```

Then, enable the constraints after the exchange.

Unless you specify `UPDATE INDEXES` (this cannot be specified for index-organized tables), the database marks `UNUSABLE` the global indexes, or all global index

partitions, on the table whose partition is being exchanged. Global indexes, or global index partitions, on the table being exchanged remain invalidated.

See Also:

- "Viewing Information About Partitioned Tables and Indexes" on page 16-70
- "Using Transportable Tablespaces: Scenarios" on page 8-54 for information about transportable tablespaces

Exchanging a Range, Hash, or List Partition

To exchange a partition of a range, hash, or list-partitioned table with a nonpartitioned table, or the reverse, use the ALTER TABLE ... EXCHANGE PARTITION statement. An example of converting a partition into a nonpartitioned table follows. In this example, table `stocks` can be range, hash, or list partitioned.

```
ALTER TABLE stocks
    EXCHANGE PARTITION p3 WITH TABLE stock_table_3;
```

Exchanging a Hash-Partitioned Table with a Range-Hash Partition

In this example, you are exchanging a whole hash-partitioned table, with all of its partitions, with the range partition of a range-hash partitioned table and all of its hash subpartitions. This is illustrated in the following example.

First, create a hash-partitioned table:

```
CREATE TABLE t1 (i NUMBER, j NUMBER)
     PARTITION BY HASH(i)
       (PARTITION p1, PARTITION p2);
```

Populate the table, then create a range-hash partitioned table as shown:

```
CREATE TABLE t2 (i NUMBER, j NUMBER)
     PARTITION BY RANGE(j)
     SUBPARTITION BY HASH(i)
        (PARTITION p1 VALUES LESS THAN (10)
            SUBPARTITION t2_pls1
            SUBPARTITION t2_pls2,
         PARTITION p2 VALUES LESS THAN (20)
            SUBPARTITION t2_p2s1
            SUBPARTITION t2_p2s2));
```

It is important that the partitioning key in table `t1` is the same as the subpartitioning key in table `t2`.

To migrate the data in t1 to t2, and validate the rows, use the following statement:

```
ALTER TABLE t2 EXCHANGE PARTITION p1 WITH TABLE t1
     WITH VALIDATION;
```

Exchanging a Subpartition of a Range-Hash Partitioned Table

Use the ALTER TABLE ... EXCHANGE PARTITION statement to convert a hash subpartition of a range-hash partitioned table into a nonpartitioned table, or the reverse. The following example converts the subpartition q3_1999_s1 of table sales into the nonpartitioned table q3_1999. Local index partitions are exchanged with corresponding indexes on q3_1999.

```
ALTER TABLE sales EXCHANGE SUBPARTITION q3_1999_s1
      WITH TABLE q3_1999 INCLUDING INDEXES;
```

Exchanging a List-Partitioned Table with a Range-List Partition

The semantics of the ALTER TABLE ... EXCHANGE PARTITION statement are the same as described previously in "Exchanging a Hash-Partitioned Table with a Range-Hash Partition". In the example shown there, the syntax of the CREATE TABLE statements would only need to be modified to create a list-partitioned table and a range-list partitioned table, respectively. The actions involved remain the same.

Exchanging a Subpartition of a Range-List Partitioned Table

The semantics of the ALTER TABLE ... EXCHANGE SUBPARTITION are the same as described previously in "Exchanging a Subpartition of a Range-Hash Partitioned Table".

Merging Partitions

Use the ALTER TABLE ... MERGE PARTITION statement to merge the contents of two partitions into one partition. The two original partitions are dropped, as are any corresponding local indexes.

You cannot use this statement for a hash-partitioned table or for hash subpartitions of a range-hash partitioned table.

If the involved partitions or subpartitions contain data, indexes may be marked UNUSABLE as explained in the following table:

Table Type	Index Behavior
Regular (Heap)	Unless you specify UPDATE INDEXES as part of the ALTER TABLE statement: ■ The database marks UNUSABLE all resulting corresponding local index partitions or subpartitions. ■ Global indexes, or all partitions of partitioned global indexes, are marked UNUSABLE and must be rebuilt.
Index-organized	■ The database marks UNUSABLE all resulting corresponding local index partitions. ■ All global indexes remain usable.

Merging Range Partitions

You are allowed to merge the contents of two adjacent range partitions into one partition. Nonadjacent range partitions cannot be merged. The resulting partition inherits the higher upper bound of the two merged partitions.

One reason for merging range partitions is to keep historical data online in larger partitions. For example, you can have daily partitions, with the oldest partition rolled up into weekly partitions, which can then be rolled up into monthly partitions, and so on.

The following scripts create an example of merging range partitions.

First, create a partitioned table and create local indexes.

```
-- Create a Table with four partitions each on its own tablespace
-- Partitioned by range on the data column.
--
CREATE TABLE four_seasons
(
        one DATE,
        two VARCHAR2(60),
        three NUMBER
)
PARTITION  BY RANGE ( one )
(
PARTITION quarter_one
   VALUES LESS THAN ( TO_DATE('01-apr-1998','dd-mon-yyyy'))
   TABLESPACE quarter_one,
PARTITION quarter_two
   VALUES LESS THAN ( TO_DATE('01-jul-1998','dd-mon-yyyy'))
   TABLESPACE quarter_two,
```

```
PARTITION quarter_three
   VALUES LESS THAN ( TO_DATE('01-oct-1998','dd-mon-yyyy'))
   TABLESPACE quarter_three,
PARTITION quarter_four
   VALUES LESS THAN ( TO_DATE('01-jan-1999','dd-mon-yyyy'))
   TABLESPACE quarter_four
);
--
-- Create local PREFIXED index on Four_Seasons
-- Prefixed because the leftmost columns of the index match the
-- Partition key
--
CREATE INDEX i_four_seasons_l ON four_seasons ( one,two )
LOCAL (
PARTITION i_quarter_one TABLESPACE i_quarter_one,
PARTITION i_quarter_two TABLESPACE i_quarter_two,
PARTITION i_quarter_three TABLESPACE i_quarter_three,
PARTITION i_quarter_four TABLESPACE i_quarter_four
);
```

Next, merge partitions.

```
--
-- Merge the first two partitions
--
ALTER TABLE four_seasons
MERGE PARTITIONS quarter_one, quarter_two INTO PARTITION quarter_two
UPDATE INDEXES;
```

If you omit the UPDATE INDEXES clause from the preceding statement, then you must rebuild the local index for the affected partition.

```
-- Rebuild index for quarter_two, which has been marked unusable
-- because it has not had all of the data from Q1 added to it.
-- Rebuilding the index will correct this.
--
ALTER TABLE four_seasons MODIFY PARTITION
quarter_two REBUILD UNUSABLE LOCAL INDEXES;
```

Merging List Partitions

When you merge list partitions, the partitions being merged can be any two partitions. They do not need to be adjacent, as for range partitions, since list partitioning does not assume any order for partitions. The resulting partition consists of all of the data from the original two partitions. If you merge a default list

partition with any other partition, the resulting partition will be the default partition.

The statement below merges two partitions of a table partitioned using the list method into a partition that inherits all of its attributes from the table-level default attributes, except for PCTFREE and MAXEXTENTS, which are specified in the statement.

```
ALTER TABLE q1_sales_by_region
   MERGE PARTITIONS q1_northcentral, q1_southcentral
   INTO PARTITION q1_central
      PCTFREE 50 STORAGE(MAXEXTENTS 20);
```

The value lists for the two original partitions were specified as:

```
PARTITION q1_northcentral VALUES ('SD','WI')
PARTITION q1_southcentral VALUES ('OK','TX')
```

The resulting `sales_west` partition value list comprises the set that represents the union of these two partition value lists, or specifically:

```
('SD','WI','OK','TX')
```

Merging Range-Hash Partitions

When you merge range-hash partitions, the subpartitions are rehashed into the number of subpartitions specified by SUBPARTITIONS *n* or the SUBPARTITION clause. If neither is included, table-level defaults are used.

Note that the inheritance of properties is different when a range-hash partition is split (discussed in "Splitting a Range-Hash Partition" on page 16-62), as opposed to when two range-hash partitions are merged. When a partition is split, the new partitions can inherit properties from the original partition since there is only one parent. However, when partitions are merged, properties must be inherited from *table level* defaults because there are two parents and the new partition cannot inherit from either at the expense of the other.

The following example merges two range-hash partitions:

```
ALTER TABLE all_seasons
   MERGE PARTITIONS quarter_1, quarter_2 INTO PARTITION quarter_2
   SUBPARTITIONS 8;
```

Merging Range-List Partitions

Partitions can be merged at the range partition level and subpartitions can be merged at the list subpartition level.

Merging Partitions in a Range-List Partitioned Table Merging range partitions in a range-list partitioned table is as described previously in "Merging Range Partitions" on page 16-46. However, when you merge two range-list partitions, the resulting new partition inherits the subpartition descriptions from the subpartition template, if one exists. If no subpartition template exists, then a single default subpartition is created for the new partition.

This following statement merges two partitions in the range-list partitioned `stripe_regional_sales` table. A subpartition template exists for the table.

```
ALTER TABLE stripe_regional_sales
   MERGE PARTITIONS q1_1999, q2_1999 INTO PARTITION q1_q2_1999
      PCTFREE 50 STORAGE(MAXEXTENTS 20);
```

Some new physical attributes are specified for this new partition while table-level defaults are inherited for those that are not specified. The new resulting partition `q1_q2_1999` inherits the high-value bound of the partition `q2_1999` and the subpartition value-list descriptions from the subpartition template description of the table.

The data in the resulting partitions consists of data from both the partitions. However, there may be cases where the database returns an error. This can occur because data may map out of the new partition when both of the following conditions exist:

- Some literal values of the merged subpartitions were not included in the subpartition template
- The subpartition template does not contain a default partition definition.

This error condition can be eliminated by always specifying a default partition in the default subpartition template.

Merging Subpartitions in a Range-List Partitioned Table You can merge the contents of any two arbitrary list subpartitions belonging to the *same* range partition. The resulting subpartition value-list descriptor includes all of the literal values in the value lists for the partitions being merged.

The following statement merges two subpartitions of a table partitioned using range-list method into a new subpartition located in tablespace `ts4`:

```
ALTER TABLE quarterly_regional_sales
   MERGE SUBPARTITIONS q1_1999_northwest, q1_1999_southwest
      INTO SUBPARTITION q1_1999_west
         TABLESPACE ts4;
```

The value lists for the original two partitions were:

- Subpartition q1_1999_northwest was described as ('WA','OR')
- Subpartition q1_1999_southwest was described as ('AZ','NM','UT')

The resulting subpartition value list comprises the set that represents the union of these two subpartition value lists:

- Subpartition q1_1999_west has a value list described as ('WA','OR','AZ','NM','UT')

The tablespace in which the resulting subpartition is located and the subpartition attributes are determined by the partition-level default attributes, except for those specified explicitly. If any of the existing subpartition names are being reused, then the new subpartition inherits the subpartition attributes of the subpartition whose name is being reused.

Modifying Default Attributes

You can modify the default attributes of a table, or for a partition of a composite partitioned table. When you modify default attributes, the new attributes affect only future partitions, or subpartitions, that are created. The default values can still be specifically overridden when creating a new partition or subpartition.

Modifying Default Attributes of a Table

You modify the default attributes that will be inherited for range, list, or hash partitions using the MODIFY DEFAULT ATTRIBUTES clause of ALTER TABLE. The following example changes the default value of PCTFREE in table emp for any new partitions that are created.

```
ALTER TABLE emp
      MODIFY DEFAULT ATTRIBUTES PCTFREE 25;
```

For hash-partitioned tables, only the TABLESPACE attribute can be modified.

Modifying Default Attributes of a Partition

To modify the default attributes inherited when creating subpartitions, use the ALTER TABLE ... MODIFY DEFAULT ATTRIBUTES FOR PARTITION. The following statement modifies the TABLESPACE in which future subpartitions of partition p1 in range-hash partitioned table emp will reside.

```
ALTER TABLE emp
     MODIFY DEFAULT ATTRIBUTES FOR PARTITION p1 TABLESPACE ts1;
```

Since all subpartitions of a range-hash partitioned table must share the same attributes, except TABLESPACE, it is the only attribute that can be changed.

Modifying Default Attributes of Index Partitions

In similar fashion to table partitions, you can alter the default attributes that will be inherited by partitions of a range-partitioned global index, or local index partitions of partitioned tables. For this you use the ALTER INDEX ... MODIFY DEFAULT ATTRIBUTES statement. Use the ALTER INDEX ... MODIFY DEFAULT ATTRIBUTES FOR PARTITION statement if you are altering default attributes to be inherited by subpartitions of a composite partitioned table.

Modifying Real Attributes of Partitions

It is possible to modify attributes of an existing partition of a table or index.

You cannot change the TABLESPACE attribute. Use ALTER TABLESPACE ... MOVE PARTITION/SUBPARTITION to move a partition or subpartition to a new tablespace.

Modifying Real Attributes for a Range or List Partition

Use the ALTER TABLE ... MODIFY PARTITION statement to modify existing attributes of a range partition or list partition. You can modify segment attributes (except TABLESPACE), or you can allocate and deallocate extents, mark local index partitions UNUSABLE, or rebuild local indexes that have been marked UNUSABLE.

If this is a range partition of a range-hash partitioned table, note the following:

- If you allocate or deallocate an extent, this action is performed for every subpartition of the specified partition.
- Likewise, changing any other attributes results in corresponding changes to those attributes of all the subpartitions for that partition. The partition level default attributes are changed as well. To avoid changing attributes of existing subpartitions, use the FOR PARTITION clause of the MODIFY DEFAULT ATTRIBUTES statement.

The following are some examples of modifying the real attributes of a partition.

This example modifies the MAXEXTENTS storage attribute for the range partition sales_q1 of table sales:

```
ALTER TABLE sales MODIFY PARTITION sales_Q1
     STORAGE (MAXEXTENTS 10);
```

All of the local index subpartitions of partition ts1 in range-hash partitioned table scubagear are marked UNUSABLE in the following example:

```
ALTER TABLE scubagear MPDIFY PARTITION ts1 UNUSABLE LOCAL INDEXES;
```

Modifying Real Attributes for a Hash Partition

You also use the ALTER TABLE ... MODIFY PARTITION statement to modify attributes of a hash partition. However, since the physical attributes of individual hash partitions must all be the same (except for TABLESPACE), you are restricted to:

- Allocating a new extent
- Deallocating an unused extent
- Marking a local index subpartition UNUSABLE
- Rebuilding local index subpartitions that are marked UNUSABLE

The following example rebuilds any unusable local index partitions associated with hash partition P1 of table dept:

```
ALTER TABLE dept MODIFY PARTITION p1
     REBUILD UNUSABLE LOCAL INDEXES;
```

Modifying Real Attributes of a Subpartition

With the MODIFY SUBPARTITION clause of ALTER TABLE you can perform the same actions as listed previously for partitions, but at the specific composite partitioned table subpartition level. For example:

```
ALTER TABLE emp MODIFY SUBPARTITION p3_s1
     REBUILD UNUSABLE LOCAL INDEXES;
```

Modifying Real Attributes of Index Partitions

The MODIFY PARTITION clause of ALTER INDEX lets you modify the real attributes of an index partition or its subpartitions. The rules are very similar to those for table partitions, but unlike the MODIFY PARTITION clause for ALTER INDEX, there is no subclause to rebuild an unusable index partition, but there is a subclause to coalesce an index partition or its subpartitions. In this context, coalesce means to merge index blocks where possible to free them for reuse.

You can also allocate or deallocate storage for a subpartition of a local index, or mark it UNUSABLE, using the MODIFY PARTITION clause.

Modifying List Partitions: Adding Values

List partitioning allows you the option of adding literal values from the defining value list.

Adding Values for a List Partition

Use the MODIFY PARTITION ... ADD VALUES clause of the ALTER TABLE statement to extend the value list of an existing partition. Literal values being added must not have been included in any other partition value list. The partition value list for any corresponding local index partition is correspondingly extended, and any global index, or global or local index partitions, remain usable.

The following statement adds a new set of state codes ('OK', 'KS') to an existing partition list.

```
ALTER TABLE sales_by_region
   MODIFY PARTITION region_south
      ADD VALUES ('OK', 'KS');
```

The existence of a default partition can have a performance impact when adding values to other partitions. This is because in order to add values to a list partition, the database must check that the values being added do not already exist in the default partition. If any of the values do exist in the default partition, an error is raised.

Note: The database executes a query to check for the existence of rows in the default partition that correspond to the literal values being added. Therefore, it is advisable to create a local prefixed index on the table. This speeds up the execution of the query and the overall operation.

You cannot add values to a default list partition.

Adding Values for a List Subpartition

This operation is essentially the same as described for "Modifying List Partitions: Adding Values", however, you use a MODIFY SUBPARTITION clause instead of the MODIFY PARTITION clause. For example, to extend the range of literal values in the value list for subpartition q1_1999_southeast use the following statement:

```
ALTER TABLE quarterly_regional_sales
   MODIFY SUBPARTITION q1_1999_southeast
```

```
   ADD VALUES ('KS');
```

Literal values being added must not have been included in any other subpartition value list within the owning partition. However, they can be duplicates of literal values in the subpartition value lists of other partitions within the table.

Modifying List Partitions: Dropping Values

List partitioning allows you the option of dropping literal values from the defining value list.

Dropping Values from a List Partition

Use the MODIFY PARTITION ... DROP VALUES clause of the ALTER TABLE statement to remove literal values from the value list of an existing partition. The statement is always executed with validation, meaning that it checks to see if any rows exist in the partition that correspond to the set of values being dropped. If any such rows are found then the database returns an error message and the operation fails. When necessary, use a DELETE statement to delete corresponding rows from the table before attempting to drop values.

Note: You cannot drop all literal values from the value list describing the partition. You must use the ALTER TABLE ... DROP PARTITION statement instead.

The partition value list for any corresponding local index partition reflects the new value list, and any global index, or global or local index partitions, remain usable.

The statement below drops a set of state codes ('OK' and 'KS') from an existing partition value list.

```
ALTER TABLE sales_by_region
   MODIFY PARTITION region_south
      DROP VALUES ('OK', 'KS');
```

Note: The database executes a query to check for the existence of rows in the partition that correspond to the literal values being dropped. Therefore, it is advisable to create a local prefixed index on the table. This speeds up the execution of the query and the overall operation.

You cannot drop values from a default list partition.

Dropping Values from a List Subpartition

This operation is essentially the same as described for "Modifying List Partitions: Dropping Values", however, you use a MODIFY SUBPARTITION clause instead of the MODIFY PARTITION clause. For example, to remove a set of literal values in the value list for subpartition q1_1999_southeast use the following statement:

```
ALTER TABLE quarterly_regional_sales
   MODIFY SUBPARTITION q1_1999_southeast
      DROP VALUES ('KS');
```

Modifying a Subpartition Template

You can modify a subpartition template of a composite partitioned table by replacing it with a new subpartition template. Any subsequent operations that use the subpartition template (such as ADD PARTITION or MERGE PARTITIONS) will now use the new subpartition template. Existing subpartitions remain unchanged.

Use the ALTER TABLE ... SET SUBPARTITION TEMPLATE statement to specify a new subpartition template. For example:

```
ALTER TABLE emp_sub_template
   SET SUBPARTITION TEMPLATE
         (SUBPARTITION e, TABLESPACE ts1,
          SUBPARTITION f, TABLESPACE ts2,
          SUBPARTITION g, TABLESPACE ts3,
          SUBPARTITION h, TABLESPACE ts4
         );
```

You can drop a subpartition template by specifying an empty list:

```
ALTER TABLE emp_sub_template
   SET SUBPARTITION TEMPLATE ( );
```

Moving Partitions

Use the MOVE PARTITION clause of the ALTER TABLE statement to:

- Re-cluster data and reduce fragmentation
- Move a partition to another tablespace
- Modify create-time attributes
- Store the data in compressed format using table compression

Typically, you can change the physical storage attributes of a partition in a single step using an ALTER TABLE / INDEX ... MODIFY PARTITION statement. However, there are some physical attributes, such as TABLESPACE, that you cannot modify using MODIFY PARTITION. In these cases, use the MOVE PARTITION clause. Modifying some other attributes, such as table compression, affects only future storage, but not existing data.

If the partition being moved contains any data, indexes may be marked UNUSABLE according to the following table:

Table Type	Index Behavior
Regular (Heap)	Unless you specify UPDATE INDEXES as part of the ALTER TABLE statement: ■ The matching partition in each local index is marked UNUSABLE. You must rebuild these index partitions after issuing MOVE PARTITION. ■ Any global indexes, or all partitions of partitioned global indexes, are marked UNUSABLE.
Index-organized	Any local or global indexes defined for the partition being moved remain usable because they are primary-key based logical rowids. However, the guess information for these rowids becomes incorrect.

Moving Table Partitions

Use the MOVE PARTITION clause to move a partition. For example, to move the most active partition to a tablespace that resides on its own disk (in order to balance I/O), not log the action, and compress the data, issue the following statement:

```
ALTER TABLE parts MOVE PARTITION depot2
     TABLESPACE ts094 NOLOGGING COMPRESS;
```

This statement always drops the old partition segment and creates a new segment, even if you do not specify a new tablespace.

If you are moving a partition of a partitioned index-organized table, you can specify the MAPPING TABLE clause as part of the MOVE PARTITION clause, and the mapping table partition will be moved to the new location along with the table partition.

Moving Subpartitions

The following statement shows how to move data in a subpartition of a table. In this example, a PARALLEL clause has also been specified.

```
ALTER TABLE scuba_gear MOVE SUBPARTITION bcd_types
     TABLESPACE tbs23 PARALLEL (DEGREE 2);
```

Moving Index Partitions

The ALTER TABLE ... MOVE PARTITION statement for regular tables, marks all partitions of a global index UNUSABLE. You can rebuild the entire index by rebuilding each partition individually using the ALTER INDEX ... REBUILD PARTITION statement. You can perform these rebuilds concurrently.

You can also simply drop the index and re-create it.

Rebuilding Index Partitions

Some reasons for rebuilding index partitions include:

- To recover space and improve performance
- To repair a damaged index partition caused by media failure
- To rebuild a local index partition after loading the underlying table partition with SQL*Loader or an import utility
- To rebuild index partitions that have been marked UNUSABLE
- To enable key compression for B-tree indexes

The following sections discuss your options for rebuilding index partitions and subpartitions.

Rebuilding Global Index Partitions

You can rebuild global index partitions in two ways:

- Rebuild each partition by issuing the ALTER INDEX ... REBUILD PARTITION statement (you can run the rebuilds concurrently).
- Drop the entire global index and re-create it. This method is more efficient because the table is scanned only once.

For most maintenance operations on partitioned tables with indexes, you can optionally avoid the need to rebuild the index by specifying UPDATE INDEXES on your DDL statement.

Rebuilding Local Index Partitions

Rebuild local indexes using either ALTER INDEX or ALTER TABLE as follows:

- ALTER INDEX ... REBUILD PARTITION / SUBPARTITION

  This statement rebuilds an index partition or subpartition unconditionally.

- ALTER TABLE ... MODIFY PARTITION / SUBPARTITION ... REBUILD UNUSABLE LOCAL INDEXES

  This statement finds all of the unusable indexes for the given table partition or subpartition and rebuilds them. It only rebuilds an index partition if it has been marked UNUSABLE.

Using ALTER INDEX to Rebuild a Partition The ALTER INDEX ... REBUILD PARTITION statement rebuilds one partition of an index. It cannot be used for composite-partitioned tables. Only real physical segments can be rebuilt with this command. When you re-create the index, you can also choose to move the partition to a new tablespace or change attributes.

For composite-partitioned tables, use ALTER INDEX ... REBUILD SUBPARTITION to rebuild a subpartition of an index. You can move the subpartition to another tablespace or specify a parallel clause. The following statement rebuilds a subpartition of a local index on a table and moves the index subpartition is another tablespace.

```
ALTER INDEX scuba
   REBUILD SUBPARTITION bcd_types
   TABLESPACE tbs23 PARALLEL (DEGREE 2);
```

Using ALTER TABLE to Rebuild an Index Partition The REBUILD UNUSABLE LOCAL INDEXES clause of ALTER TABLE ... MODIFY PARTITION does not allow you to specify any new attributes for the rebuilt index partition. The following example finds and rebuilds any unusable local index partitions for table scubagear, partition p1.

```
ALTER TABLE scubagear
   MODIFY PARTITION p1 REBUILD UNUSABLE LOCAL INDEXES;
```

There is a corresponding ALTER TABLE ... MODIFY SUBPARTITION clause for rebuilding unusable local index subpartitions.

Renaming Partitions

It is possible to rename partitions and subpartitions of both tables and indexes. One reason for renaming a partition might be to assign a meaningful name, as opposed to a default system name that was assigned to the partition in another maintenance operation.

Renaming a Table Partition

Rename a range, hash, or list partition, using the ALTER TABLE ... RENAME PARTITION statement. For example:

```
ALTER TABLE scubagear RENAME PARTITION sys_p636 TO tanks;
```

Renaming a Table Subpartition

Likewise, you can assign new names to subpartitions of a table. In this case you would use the ALTER TABLE ... RENAME PARTITION syntax.

Renaming Index Partitions

Index partitions and subpartitions can be renamed in similar fashion, but the ALTER INDEX syntax is used.

Renaming an Index Partition Use the ALTER INDEX ... RENAME PARTITION statement to rename an index partition.

Renaming an Index Subpartition This next statement simply shows how to rename a subpartition that has a system generated name that was a consequence of adding a partition to an underlying table:

```
ALTER INDEX scuba RENAME SUBPARTITION sys_subp3254 TO bcd_types;
```

Splitting Partitions

The SPLIT PARTITION clause of the ALTER TABLE or ALTER INDEX statement is used to redistribute the contents of a partition into two new partitions. Consider doing this when a partition becomes too large and causes backup, recovery, or maintenance operations to take a long time to complete. You can also use the SPLIT PARTITION clause to redistribute the I/O load.

This clause cannot be used for hash partitions or subpartitions.

If the partition you are splitting contains any data, indexes may be marked UNUSABLE as explained in the following table:

Table Type	Index Behavior
Regular (Heap)	Unless you specify UPDATE INDEXES as part of the ALTER TABLE statement: ■ The database marks UNUSABLE the new partitions (there are two) in each local index. ■ Any global indexes, or all partitions of partitioned global indexes, are marked UNUSABLE and must be rebuilt.
Index-organized	■ The database marks UNUSABLE the new partitions (there are two) in each local index. ■ All global indexes remain usable.

Splitting a Partition of a Range-Partitioned Table

You split a range partition using the ALTER TABLE ... SPLIT PARTITION statement. You specify a value of the partitioning key column within the range of the partition at which to split the partition. The first of the resulting two new partitions includes all rows in the original partition whose partitioning key column values map lower that the specified value. The second partition contains all rows whose partitioning key column values map greater than or equal to the specified value.

You can optionally specify new attributes for the two partitions resulting from the split. If there are local indexes defined on the table, this statement also splits the matching partition in each local index.

In the following example `fee_katy` is a partition in the table `vet_cats`, which has a local index, `jaf1`. There is also a global index, `vet` on the table. `vet` contains two partitions, `vet_parta`, and `vet_partb`.

To split the partition `fee_katy`, and rebuild the index partitions, issue the following statements:

```
ALTER TABLE vet_cats SPLIT PARTITION
      fee_katy at (100) INTO ( PARTITION
      fee_katy1 ..., PARTITION fee_katy2 ...);
ALTER INDEX JAF1 REBUILD PARTITION fee_katy1;
ALTER INDEX JAF1 REBUILD PARTITION fee_katy2;
ALTER INDEX VET REBUILD PARTITION vet_parta;
ALTER INDEX VET REBUILD PARTITION vet_partb;
```

Note: If you do not specify new partition names, the database assigns names of the form SYS_P*n*. You can examine the data dictionary to locate the names assigned to the new local index partitions. You may want to rename them. Any attributes you do not specify are inherited from the original partition.

Splitting a Partition of a List-Partitioned Table

You split a list partition using the ALTER TABLE ... SPLIT PARTITION statement. The SPLIT PARTITION clause lets you specify a value list of literal values that define a partition into which rows with corresponding partitioning key values are inserted. The remaining rows of the original partition are inserted into a second partition whole value list is the remaining values from the original partition.

You can optionally specify new attributes for the two partitions resulting from the split.

The following statement splits the partition `region_east` into two partitions:

```
ALTER TABLE sales_by_region
   SPLIT PARTITION region_east VALUES ('CT', 'VA', 'MD')
   INTO
    ( PARTITION region_east_1
         PCTFREE 25 TABLESPACE tbs2,
      PARTITION region_east_2
        STORAGE (NEXT 2M PCTINCREASE 25))
   PARALLEL 5;
```

The literal-value list for the original `region_east` partition was specified as:

```
PARTITION region_east VALUES ('MA','NY','CT','NH','ME','MD','VA','PA','NJ')
```

The two new partitions are:

- `region_east_1` with a literal-value list of `('CT','VA','MD')`
- `region_east_2` inheriting the remaining literal-value list of `('NY','NH','ME','VA','PA','NJ')`

The individual partitions have new physical attributes specified at the partition level. The operation is executed with parallelism of degree 5.

You can split a default list partition just like you split any other list partition. This is also the only means of adding a partition to list-partitioned table that contains a default partition. When you split the default partition, you create a new partition

defined by the values that you specify, and a second partition that remains the default partition.

The following example splits the default partition of `sales_by_region`, thereby creating a new partition.

```
ALTER TABLE sales_by_region
   SPLIT PARTITION region_unknown VALUES ('MT', 'WY', 'ID')
   INTO
    ( PARTITION region_wildwest,
      PARTITION region_unknown);
```

Splitting a Range-Hash Partition

This is the opposite of merging range-hash partitions. When you split range-hash partitions, the new subpartitions are rehashed into either the number of subpartitions specified in a SUBPARTITIONS or SUBPARTITION clause. Or, if no such clause is included, the new partitions inherit the number of subpartitions (and tablespaces) from the partition being split.

Note that the inheritance of properties is different when a range-hash partition is split, verses when two range-hash partitions are merged. When a partition is split, the new partitions can inherit properties from the original partition since there is only one parent. However, when partitions are merged, properties must be inherited from *table level* defaults because there are two parents and the new partition cannot inherit from either at the expense of the other.

The following example splits a range-hash partition:

```
ALTER TABLE all_seasons SPLIT PARTITION quarter_1
     AT (TO_DATE('16-dec-1997','dd-mon-yyyy'))
     INTO (PARTITION q1_1997_1 SUBPARTITIONS 4 STORE IN (ts1,ts3),
           PARTITION q1_1997_2);
```

Splitting Partitions in a Range-List Partitioned Table

Partitions can be split at both the range partition level and at the list subpartition level.

Splitting a Range-List Partition Splitting a range partition of a range-list partitioned table is similar to what is described in "Splitting a Partition of a Range-Partitioned Table" on page 16-60. No subpartition literal value list can be specified for either of the new partitions. The new partitions inherit the subpartition descriptions from the original partition being split.

The following example splits the q1_1999 partition of the quarterly_regional_sales table:

```
ALTER TABLE quarterly_regional_sales SPLIT PARTITION q1_1999
   AT (to_date('15-Feb-1999','dd-mon-yyyy'))
   INTO ( PARTITION q1_1999_jan_feb
             PCTFREE 25 TABLESPACE ts1,
          PARTITION q1_1999_feb_mar
             STORAGE (NEXT 2M PCTINCREASE 25) TABLESPACE ts2)
   PARALLEL 5;
```

This operation splits the partition q1_1999 into two resulting partitions: q1_1999_jan_feb and q1_1999_feb_mar. Both partitions inherit their subpartition descriptions from the original partition. The individual partitions have new physical attributes, including tablespaces, specified at the partition level. These new attributes become the default attributes of the new partitions. This operation is run with parallelism of degree 5.

The ALTER TABLE ... SPLIT PARTITION statement provides no means of specifically naming subpartitions resulting from the split of a partition in a composite partitioned table. However, for those subpartitions in the parent partition with names of the form *partition name_subpartition name*, the database generates corresponding names in the newly created subpartitions using the new partition names. All other subpartitions are assigned system generated names of the form SYS_SUBP*n*. System generated names are also assigned for the subpartitions of any partition resulting from the split for which a name is not specified. Unnamed partitions are assigned a system generated partition name of the form SYS_P*n*.

The following query displays the subpartition names resulting from the previous split partition operation on table quarterly_regional_sales. It also reflects the results of other operations performed on this table in preceding sections of this chapter since its creation in "When to Use Composite Range-List Partitioning" on page 16-8.

```
SELECT PARTITION_NAME, SUBPARTITION_NAME, TABLESPACE_NAME
  FROM DBA_TAB_SUBPARTITIONS
  WHERE TABLE_NAME='QUARTERLY_REGIONAL_SALES'
  ORDER BY PARTITION_NAME;

PARTITION_NAME       SUBPARTITION_NAME              TABLESPACE_NAME
-------------------- ------------------------------ --------------
Q1_1999_FEB_MAR      Q1_1999_FEB_MAR_WEST           TS2
Q1_1999_FEB_MAR      Q1_1999_FEB_MAR_NORTHEAST      TS2
Q1_1999_FEB_MAR      Q1_1999_FEB_MAR_SOUTHEAST      TS2
Q1_1999_FEB_MAR      Q1_1999_FEB_MAR_NORTHCENTRAL   TS2
```

```
Q1_1999_FEB_MAR      Q1_1999_FEB_MAR_SOUTHCENTRAL   TS2
Q1_1999_FEB_MAR      Q1_1999_FEB_MAR_SOUTH          TS2
Q1_1999_JAN_FEB      Q1_1999_JAN_FEB_WEST           TS1
Q1_1999_JAN_FEB      Q1_1999_JAN_FEB_NORTHEAST      TS1
Q1_1999_JAN_FEB      Q1_1999_JAN_FEB_SOUTHEAST      TS1
Q1_1999_JAN_FEB      Q1_1999_JAN_FEB_NORTHCENTRAL   TS1
Q1_1999_JAN_FEB      Q1_1999_JAN_FEB_SOUTHCENTRAL   TS1
Q1_1999_JAN_FEB      Q1_1999_JAN_FEB_SOUTH          TS1
Q1_2000              Q1_2000_NORTHWEST              TS3
Q1_2000              Q1_2000_SOUTHWEST              TS3
Q1_2000              Q1_2000_NORTHEAST              TS3
Q1_2000              Q1_2000_SOUTHEAST              TS3
Q1_2000              Q1_2000_NORTHCENTRAL           TS3
Q1_2000              Q1_2000_SOUTHCENTRAL           TS3
Q2_1999              Q2_1999_NORTHWEST              TS4
Q2_1999              Q2_1999_SOUTHWEST              TS4
Q2_1999              Q2_1999_NORTHEAST              TS4
Q2_1999              Q2_1999_SOUTHEAST              TS4
Q2_1999              Q2_1999_NORTHCENTRAL           TS4
Q2_1999              Q2_1999_SOUTHCENTRAL           TS4
Q3_1999              Q3_1999_NORTHWEST              TS4
Q3_1999              Q3_1999_SOUTHWEST              TS4
Q3_1999              Q3_1999_NORTHEAST              TS4
Q3_1999              Q3_1999_SOUTHEAST              TS4
Q3_1999              Q3_1999_NORTHCENTRAL           TS4
Q3_1999              Q3_1999_SOUTHCENTRAL           TS4
Q4_1999              Q4_1999_NORTHWEST              TS4
Q4_1999              Q4_1999_SOUTHWEST              TS4
Q4_1999              Q4_1999_NORTHEAST              TS4
Q4_1999              Q4_1999_SOUTHEAST              TS4
Q4_1999              Q4_1999_NORTHCENTRAL           TS4
Q4_1999              Q4_1999_SOUTHCENTRAL           TS4

36 rows selected.
```

Splitting a Range-List Subpartition Splitting a list subpartition of a range-list partitioned table is similar to what is described in "Splitting a Partition of a List-Partitioned Table" on page 16-61, but the syntax is that of `SUBPARTITION` rather than `PARTITION`. For example, the following statement splits a subpartition of the `quarterly_regional_sales` table:

```
ALTER TABLE quarterly_regional_sales SPLIT SUBPARTITION q2_1999_southwest
   VALUES ('UT') INTO
      ( SUBPARTITION q2_1999_utah
           TABLESPACE ts2,
```

```
      SUBPARTITION q2_1999_southwest
         TABLESPACE ts3
   )
PARALLEL;
```

This operation splits the subpartition `q2_1999_southwest` into two subpartitions:

- `q2_1999_utah` with literal-value list of `('UT')`
- `q2_1999_southwest` which inherits the remaining literal-value list of `('AZ','NM')`

The individual subpartitions have new physical attributes that are inherited from the subpartition being split.

Splitting Index Partitions

You cannot explicitly split a partition in a local index. A local index partition is split only when you split a partition in the underlying table. However, you can split a global index partition as is done in the following example:

```
ALTER INDEX quon1 SPLIT
    PARTITION canada AT ( 100 ) INTO
    PARTITION canada1 ..., PARTITION canada2 ...);
ALTER INDEX quon1 REBUILD PARTITION canada1;
ALTER INDEX quon1 REBUILD PARTITION canada2;
```

The index being split can contain index data, and the resulting partitions do not require rebuilding, unless the original partition was previously marked `UNUSABLE`.

Optimizing SPLIT PARTITION and SPLIT SUBPARTITION Operations

Oracle Database implements a `SPLIT PARTITION` operation by creating two new partitions and redistributing the rows from the partition being split into the two new partitions. This is an expensive operation because it is necessary to scan all the rows of the partition being split and then insert them one-by-one into the new partitions. Further if you do not use the `UPDATE INDEXES` clause, both local and global indexes also require rebuilding.

Sometimes after a split operation, one of the new partitions contains all of the rows from the partition being split, while the other partition contains no rows. This is often the case when splitting the first partition of a table. The database can detect such situations in heap-organized partitioned tables and can optimize the split operation. This optimization results in a fast split operation that behaves like an add

partition operation. The fast split optimization does not apply to partitioned index-organized tables.

Specifically, the database can optimize and speed up SPLIT PARTITION operations if two conditions are met:

- One of the two resulting partitions must be empty.
- The nonempty resulting partition must have storage characteristics identical to those of the partition being split. Specifically:
  - If the partition being split is composite, then the storage characteristics of each subpartition in the new nonempty resulting partition must be identical to those of the subpartitions of the partition being split.
  - If the partition being split contains a LOB column, then the storage characteristics of each LOB (sub)partition in the new nonempty resulting partition must be identical to those of the LOB (sub)partitions of the partition being split.

If both of these conditions are met after the split, then all global indexes remain usable, even if you did not specify the UPDATE INDEXES clause. Local index (sub)partitions associated with both resulting partitions remain usable if they were usable before the split. Local index (sub)partition(s) corresponding to the nonempty resulting partition will be identical to the local index (sub)partition(s) of the partition that was split.

The same optimization holds for SPLIT SUBPARTITION operations.

Truncating Partitions

Use the ALTER TABLE ... TRUNCATE PARTITION statement to remove all rows from a table partition. Truncating a partition is similar to dropping a partition, except that the partition is emptied of its data, but not physically dropped.

You cannot truncate an index partition. However, if local indexes are defined for the table, the ALTER TABLE ... TRUNCATE PARTITION statement truncates the matching partition in each local index. Unless you specify UPDATE INDEXES (cannot be specified for index-organized tables), any global indexes are marked UNUSABLE and must be rebuilt.

Truncating a Table Partition

Use the ALTER TABLE ... TRUNCATE PARTITION statement to remove all rows from a table partition, with or without reclaiming space.

Truncating Table Partitions Containing Data and Global Indexes If the partition contains data and global indexes, use one of the following methods to truncate the table partition.

Method 1

Leave the global indexes in place during the ALTER TABLE ... TRUNCATE PARTITION statement. In this example, table `sales` has a global index `sales_area_ix`, which is rebuilt.

```
ALTER TABLE sales TRUNCATE PARTITION dec98;
ALTER INDEX sales_area_ix REBUILD;
```

This method is most appropriate for large tables where the partition being truncated contains a significant percentage of the total data in the table.

Method 2

Issue the DELETE statement to delete all rows from the partition before you issue the ALTER TABLE ... TRUNCATE PARTITION statement. The DELETE statement updates the global indexes, and also fires triggers and generates redo and undo logs.

For example, to truncate the first partition, which has a partition bound of 10000, issue the following statements:

```
DELETE FROM sales WHERE TRANSID < 10000;
ALTER TABLE sales TRUNCATE PARTITION dec98;
```

This method is most appropriate for small tables, or for large tables when the partition being truncated contains a small percentage of the total data in the table.

Method 3

Specify UPDATE INDEXES in the ALTER TABLE statement. This causes the global index to be truncated at the time the partition is truncated.

```
ALTER TABLE sales TRUNCATE PARTITION dec98
     UPDATE INDEXES;
```

Truncating a Partition Containing Data and Referential Integrity Constraints If a partition contains data and has referential integrity constraints, choose either of the following methods to truncate the table partition.

Method 1

Disable the integrity constraints, issue the ALTER TABLE ... TRUNCATE PARTITION statement, then reenable the integrity constraints:

```
ALTER TABLE sales
    DISABLE CONSTRAINT dname_sales1;
ALTER TABLE sales TRUNCATE PARTITTION dec94;
ALTER TABLE sales
    ENABLE CONSTRAINT dname_sales1;
```

This method is most appropriate for large tables where the partition being truncated contains a significant percentage of the total data in the table.

Method 2

Issue the DELETE statement to delete all rows from the partition before you issue the ALTER TABLE ... TRUNCATE PARTITION statement. The DELETE statement enforces referential integrity constraints, and also fires triggers and generates redo and undo log.

> **Note:** You can substantially reduce the amount of logging by setting the NOLOGGING attribute (using ALTER TABLE ... TRUNCATE PARTITION ... NOLOGGING) for the partition before deleting all of its rows.

```
DELETE FROM sales WHERE TRANSID < 10000;
ALTER TABLE sales TRUNCATE PARTITION dec94;
```

This method is most appropriate for small tables, or for large tables when the partition being truncated contains a small percentage of the total data in the table.

Truncating a Subpartition

You use the ALTER TABLE ... TRUNCATE SUBPARTITION statement to remove all rows from a subpartition of a composite partitioned table. Corresponding local index subpartitions are also truncated.

The following statement shows how to truncate data in a subpartition of a table. In this example, the space occupied by the deleted rows is made available for use by other schema objects in the tablespace.

```
ALTER TABLE diving
   TRUNCATE SUBPARTITION us_locations
```

```
DROP STORAGE;
```

Partitioned Tables and Indexes Example

This section presents an example of moving the time window in a historical table.

A **historical** table describes the business transactions of an enterprise over intervals of time. Historical tables can be **base** tables, which contain base information; for example, sales, checks, and orders. Historical tables can also be **rollup** tables, which contain summary information derived from the base information using operations such as GROUP BY, AVERAGE, or COUNT.

The time interval in a historical table is often a rolling window. DBAs periodically delete sets of rows that describe the oldest transactions, and in turn allocate space for sets of rows that describe the most recent transactions. For example, at the close of business on April 30, 1995, the DBA deletes the rows (and supporting index entries) that describe transactions from April 1994, and allocates space for the April 1995 transactions.

Now consider a specific example. You have a table, `order`, which contains 13 months of transactions: a year of historical data in addition to orders for the current month. There is one partition for each month. These monthly partitions are named `order_`*`yymm`*, as are the tablespaces in which they reside.

The `order` table contains two local indexes, `order_ix_onum`, which is a local, prefixed, unique index on the order number, and `order_ix_supp`, which is a local, nonprefixed index on the supplier number. The local index partitions are named with suffixes that match the underlying table. There is also a global unique index, `order_ix_cust`, for the customer name. `order_ix_cust` contains three partitions, one for each third of the alphabet. So on October 31, 1994, change the time window on `order` as follows:

1. Back up the data for the oldest time interval.

   ```
   ALTER TABLESPACE order_9310 BEGIN BACKUP;
   ...
   ALTER TABLESPACE order_9310 END BACKUP;
   ```

2. Drop the partition for the oldest time interval.

   ```
   ALTER TABLE order DROP PARTITION order_9310;
   ```

3. Add the partition to the most recent time interval.

   ```
   ALTER TABLE order ADD PARTITION order_9411;
   ```

4. Re-create the global index partitions.

```
ALTER INDEX order_ix_cust REBUILD PARTITION order_ix_cust_AH;
ALTER INDEX order_ix_cust REBUILD PARTITION order_ix_cust_IP;
ALTER INDEX order_ix_cust REBUILD PARTITION order_ix_cust_QZ;
```

Ordinarily, the database acquires sufficient locks to ensure that no operation (DML, DDL, or utility) interferes with an individual DDL statement, such as ALTER TABLE ... DROP PARTITION. However, if the partition maintenance operation requires several steps, it is the database administrator's responsibility to ensure that applications (or other maintenance operations) do not interfere with the multistep operation in progress. Some methods for doing this are:

- Bring down all user-level applications during a well-defined batch window.
- Ensure that no one is able to access table `order` by revoking access privileges from a role that is used in all applications.

Viewing Information About Partitioned Tables and Indexes

The following views display information specific to partitioned tables and indexes:

View	Description
DBA_PART_TABLES ALL_PART_TABLES USER_PART_TABLES	DBA view displays partitioning information for all partitioned tables in the database. ALL view displays partitioning information for all partitioned tables accessible to the user. USER view is restricted to partitioning information for partitioned tables owned by the user.
DBA_TAB_PARTITIONS ALL_TAB_PARTITIONS USER_TAB_PARTITIONS	Display partition-level partitioning information, partition storage parameters, and partition statistics generated by the DBMS_STATS package or the ANALYZE statement.
DBA_TAB_SUBPARTITIONS ALL_TAB_SUBPARTITIONS USER_TAB_SUBPARTITIONS	Display subpartition-level partitioning information, subpartition storage parameters, and subpartition statistics generated by the DBMS_STATS package or the ANALYZE statement.
DBA_PART_KEY_COLUMNS ALL_PART_KEY_COLUMNS USER_PART_KEY_COLUMNS	Display the partitioning key columns for partitioned tables.

View	Description
DBA_SUBPART_KEY_COLUMNS ALL_SUBPART_KEY_COLUMNS USER_SUBPART_KEY_COLUMNS	Display the subpartitioning key columns for composite-partitioned tables (and local indexes on composite-partitioned tables).
DBA_PART_COL_STATISTICS ALL_PART_COL_STATISTICS USER_PART_COL_STATISTICS	Display column statistics and histogram information for the partitions of tables.
DBA_SUBPART_COL_STATISTICS ALL_SUBPART_COL_STATISTICS USER_SUBPART_COL_STATISTICS	Display column statistics and histogram information for subpartitions of tables.
DBA_PART_HISTOGRAMS ALL_PART_HISTOGRAMS USER_PART_HISTOGRAMS	Display the histogram data (end-points for each histogram) for histograms on table partitions.
DBA_SUBPART_HISTOGRAMS ALL_SUBPART_HISTOGRAMS USER_SUBPART_HISTOGRAMS	Display the histogram data (end-points for each histogram) for histograms on table subpartitions.
DBA_PART_INDEXES ALL_PART_INDEXES USER_PART_INDEXES	Display partitioning information for partitioned indexes.
DBA_IND_PARTITIONS ALL_IND_PARTITIONS USER_IND_PARTITIONS	Display the following for index partitions: partition-level partitioning information, storage parameters for the partition, statistics collected by the DBMS_STATS package or the ANALYZE statement.
DBA_IND_SUBPARTITIONS ALL_IND_SUBPARTITIONS USER_IND_SUBPARTITIONS	Display the following information for index subpartitions: partition-level partitioning information, storage parameters for the partition, statistics collected by the DBMS_STATS package or the ANALYZE statement.
DBA_SUBPARTITION_TEMPLATES ALL_SUBPARTITION_TEMPLATES USER_SUBPARTITION_TEMPLATES	Display information about existing subpartition templates.

See Also:

- *Oracle Database Reference* for complete descriptions of these views
- *Oracle Database Performance Tuning Guide* and *Oracle Database Performance Tuning Guide* for information about histograms and generating statistics for tables
- "Analyzing Tables, Indexes, and Clusters" on page 20-2

17

Managing Clusters

This chapter describes aspects of managing clusters. It contains the following topics relating to the management of indexed clusters, clustered tables, and cluster indexes:

- About Clusters
- Guidelines for Managing Clusters
- Creating Clusters
- Altering Clusters
- Dropping Clusters
- Viewing Information About Clusters

About Clusters

A **cluster** provides an optional method of storing table data. A cluster is made up of a group of tables that share the same data blocks. The tables are grouped together because they share common columns and are often used together. For example, the `emp` and `dept` table share the `deptno` column. When you cluster the `emp` and `dept` tables (see Figure 17–1), Oracle Database physically stores all rows for each department from both the `emp` and `dept` tables in the same data blocks.

Because clusters store related rows of different tables together in the same data blocks, properly used clusters offer two primary benefits:

- Disk I/O is reduced and access time improves for joins of clustered tables.
- The **cluster key** is the column, or group of columns, that the clustered tables have in common. You specify the columns of the cluster key when creating the cluster. You subsequently specify the same columns when creating every table

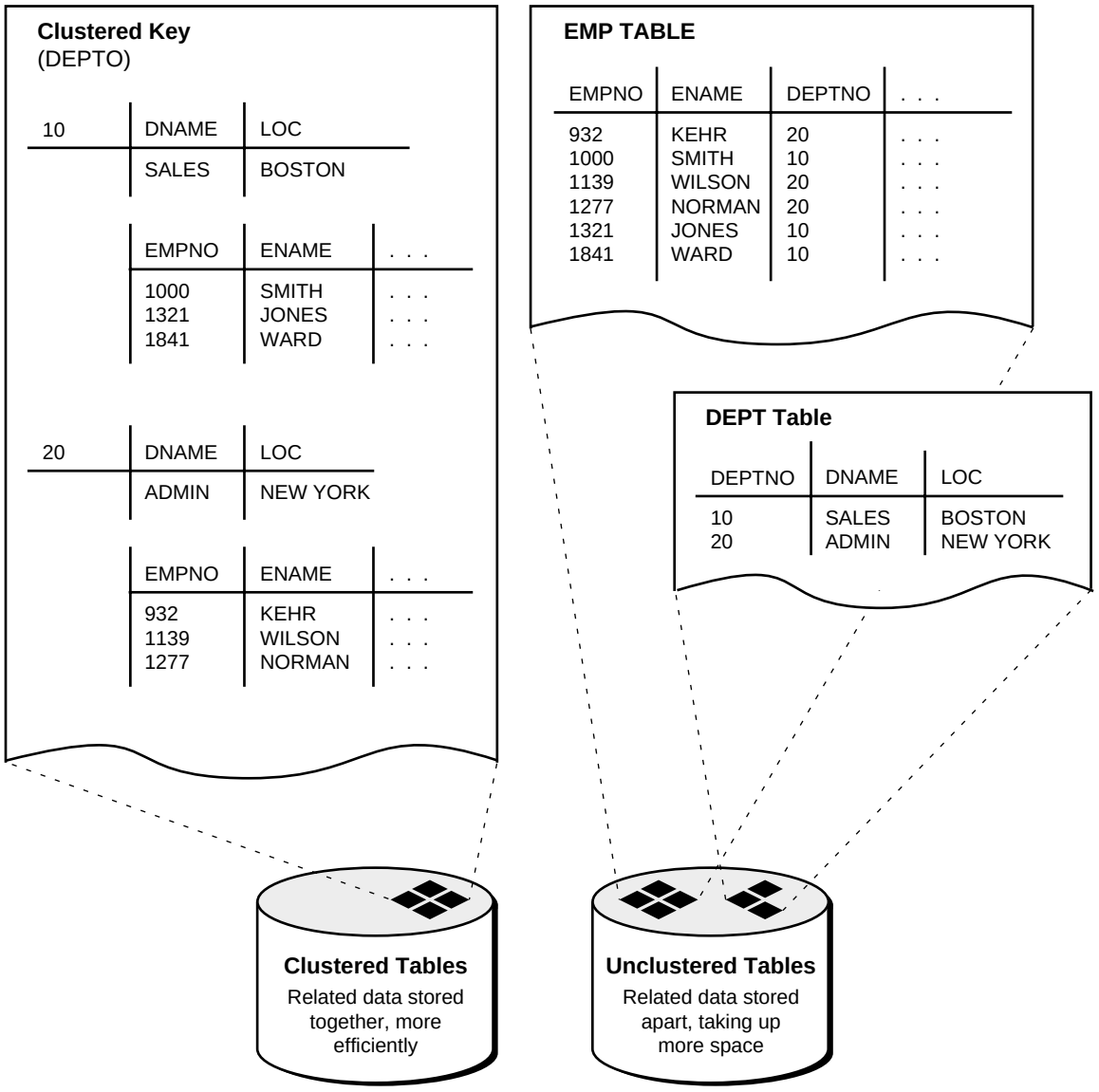
added to the cluster. Each cluster key value is stored only once each in the cluster and the cluster index, no matter how many rows of different tables contain the value.

Therefore, less storage might be required to store related table and index data in a cluster than is necessary in nonclustered table format. For example, in Figure 17–1, notice how each cluster key (each `deptno`) is stored just once for many rows that contain the same value in both the `emp` and `dept` tables.

After creating a cluster, you can create tables in the cluster. However, before any rows can be inserted into the clustered tables, a cluster index must be created. Using clusters does not affect the creation of additional indexes on the clustered tables; they can be created and dropped as usual.

You should not use clusters for tables that are frequently accessed individually.

Figure 17–1 Clustered Table Data



See Also:

- Chapter 18, "Managing Hash Clusters" for a description of another type of cluster: a hash cluster
- Chapter 13, "Managing Space for Schema Objects" is recommended reading before attempting tasks described in this chapter

Guidelines for Managing Clusters

The following sections describe guidelines to consider when managing clusters, and contains the following topics:

- Choose Appropriate Tables for the Cluster
- Choose Appropriate Columns for the Cluster Key
- Specify Data Block Space Use
- Specify the Space Required by an Average Cluster Key and Its Associated Rows
- Specify the Location of Each Cluster and Cluster Index Rows
- Estimate Cluster Size and Set Storage Parameters

See Also:

- *Oracle Database Concepts* for more information about clusters
- *Oracle Database Performance Tuning Guide* for guidelines on when to use clusters

Choose Appropriate Tables for the Cluster

Use clusters for tables for which the following conditions are true:

- The tables are primarily queried--that is, tables that are *not* predominantly inserted into or updated.
- Records from the tables are frequently queried together or joined.

Choose Appropriate Columns for the Cluster Key

Choose cluster key columns carefully. If multiple columns are used in queries that join the tables, make the cluster key a composite key. In general, the characteristics that indicate a good cluster index are the same as those for any index. For

information about characteristics of a good index, see "Guidelines for Managing Indexes" on page 15-2.

A good cluster key has enough unique values so that the group of rows corresponding to each key value fills approximately one data block. Having too few rows for each cluster key value can waste space and result in negligible performance gains. Cluster keys that are so specific that only a few rows share a common value can cause wasted space in blocks, unless a small SIZE was specified at cluster creation time (see "Specify the Space Required by an Average Cluster Key and Its Associated Rows" on page 17-5).

Too many rows for each cluster key value can cause extra searching to find rows for that key. Cluster keys on values that are too general (for example, male and female) result in excessive searching and can result in worse performance than with no clustering.

A cluster index cannot be unique or include a column defined as long.

Specify Data Block Space Use

By specifying the PCTFREE and PCTUSED parameters during the creation of a cluster, you can affect the space utilization and amount of space reserved for updates to the current rows in the data blocks of a cluster data segment. PCTFREE and PCTUSED parameters specified for tables created in a cluster are ignored; clustered tables automatically use the settings specified for the cluster.

> **See Also:** "Managing Space in Data Blocks" on page 13-1 for information about setting the PCTFREE and PCTUSED parameters

Specify the Space Required by an Average Cluster Key and Its Associated Rows

The CREATE CLUSTER statement has an optional clause, SIZE, which is the estimated number of bytes required by an average cluster key and its associated rows. The database uses the SIZE parameter when performing the following tasks:

- Estimating the number of cluster keys (and associated rows) that can fit in a clustered data block
- Limiting the number of cluster keys placed in a clustered data block. This maximizes the storage efficiency of keys within a cluster.

SIZE does not limit the space that can be used by a given cluster key. For example, if SIZE is set such that two cluster keys can fit in one data block, any amount of the available data block space can still be used by either of the cluster keys.

By default, the database stores only one cluster key and its associated rows in each data block of the cluster data segment. Although block size can vary from one operating system to the next, the rule of one key for each block is maintained as clustered tables are imported to other databases on other machines.

If all the rows for a given cluster key value cannot fit in one block, the blocks are chained together to speed access to all the values with the given key. The cluster index points to the beginning of the chain of blocks, each of which contains the cluster key value and associated rows. If the cluster `SIZE` is such that more than one key fits in a block, blocks can belong to more than one chain.

Specify the Location of Each Cluster and Cluster Index Rows

If you have the proper privileges and tablespace quota, you can create a new cluster and the associated cluster index in any tablespace that is currently online. Always specify the `TABLESPACE` clause in a `CREATE CLUSTER/INDEX` statement to identify the tablespace to store the new cluster or index.

The cluster and its cluster index can be created in different tablespaces. In fact, creating a cluster and its index in different tablespaces that are stored on different storage devices allows table data and index data to be retrieved simultaneously with minimal disk contention.

Estimate Cluster Size and Set Storage Parameters

The following are benefits of estimating cluster size before creating the cluster:

- You can use the combined estimated size of clusters, along with estimates for indexes and redo log files, to determine the amount of disk space that is required to hold an intended database. From these estimates, you can make correct hardware purchases and other decisions.
- You can use the estimated size of an individual cluster to better manage the disk space that the cluster will use. When a cluster is created, you can set appropriate storage parameters and improve I/O performance of applications that use the cluster.

Whether or not you estimate table size before creation, you can explicitly set storage parameters when creating each nonclustered table. Any storage parameter that you do not explicitly set when creating or subsequently altering a table automatically uses the corresponding default storage parameter set for the tablespace in which the table resides. Clustered tables also automatically use the storage parameters of the cluster.

Creating Clusters

To create a cluster in your schema, you must have the CREATE CLUSTER system privilege and a quota for the tablespace intended to contain the cluster or the UNLIMITED TABLESPACE system privilege.

To create a cluster in another user's schema you must have the CREATE ANY CLUSTER system privilege, and the owner must have a quota for the tablespace intended to contain the cluster or the UNLIMITED TABLESPACE system privilege.

You create a cluster using the CREATE CLUSTER statement. The following statement creates a cluster named emp_dept, which stores the emp and dept tables, clustered by the deptno column:

```
CREATE CLUSTER emp_dept (deptno NUMBER(3))
   PCTUSED 80
   PCTFREE 5
   SIZE 600
   TABLESPACE users
   STORAGE (INITIAL 200K
      NEXT 300K
      MINEXTENTS 2
      MAXEXTENTS 20
      PCTINCREASE 33);
```

If no INDEX keyword is specified, as is true in this example, an index cluster is created by default. You can also create a HASH cluster, when hash parameters (HASHKEYS, HASH IS, or SINGLE TABLE HASHKEYS) are specified. Hash clusters are described in Chapter 18, "Managing Hash Clusters".

Creating Clustered Tables

To create a table in a cluster, you must have either the CREATE TABLE or CREATE ANY TABLE system privilege. You do not need a tablespace quota or the UNLIMITED TABLESPACE system privilege to create a table in a cluster.

You create a table in a cluster using the CREATE TABLE statement with the CLUSTER clause. The emp and dept tables can be created in the emp_dept cluster using the following statements:

```
CREATE TABLE emp (
   empno NUMBER(5) PRIMARY KEY,
   ename VARCHAR2(15) NOT NULL,
   . . .
   deptno NUMBER(3) REFERENCES dept)
```

```
   CLUSTER emp_dept (deptno);

CREATE TABLE dept (
   deptno NUMBER(3) PRIMARY KEY, . . . )
   CLUSTER emp_dept (deptno);
```

Note: You can specify the schema for a clustered table in the CREATE TABLE statement. A clustered table can be in a different schema than the schema containing the cluster. Also, the names of the columns are not required to match, but their structure must match.

See Also: *Oracle Database SQL Reference* for syntax of the CREATE TABLE statement for creating cluster tables

Creating Cluster Indexes

To create a cluster index, one of the following conditions must be true:

- Your schema contains the cluster.
- You have the CREATE ANY INDEX system privilege.

In either case, you must also have either a quota for the tablespace intended to contain the cluster index, or the UNLIMITED TABLESPACE system privilege.

A cluster index must be created before any rows can be inserted into any clustered table. The following statement creates a cluster index for the emp_dept cluster:

```
CREATE INDEX emp_dept_index
   ON CLUSTER emp_dept
   TABLESPACE users
   STORAGE (INITIAL 50K
      NEXT 50K
      MINEXTENTS 2
      MAXEXTENTS 10
      PCTINCREASE 33)
   PCTFREE 5;
```

The cluster index clause (ON CLUSTER) identifies the cluster, emp_dept, for which the cluster index is being created. The statement also explicitly specifies several storage settings for the cluster and cluster index.

See Also: *Oracle Database SQL Reference* for syntax of the CREATE INDEX statement for creating cluster indexes

Altering Clusters

To alter a cluster, your schema must contain the cluster or you must have the ALTER ANY CLUSTER system privilege. You can alter an existing cluster to change the following settings:

- Physical attributes (PCTFREE, PCTUSED, INITRANS, and storage characteristics)
- The average amount of space required to store all the rows for a cluster key value (SIZE)
- The default degree of parallelism

Additionally, you can explicitly allocate a new extent for the cluster, or deallocate any unused extents at the end of the cluster. The database dynamically allocates additional extents for the data segment of a cluster as required. In some circumstances, however, you might want to explicitly allocate an additional extent for a cluster. For example, when using Real Application Clusters, you can allocate an extent of a cluster explicitly for a specific instance. You allocate a new extent for a cluster using the ALTER CLUSTER statement with the ALLOCATE EXTENT clause.

When you alter data block space usage parameters (PCTFREE and PCTUSED) or the cluster size parameter (SIZE) of a cluster, the new settings apply to all data blocks used by the cluster, including blocks already allocated and blocks subsequently allocated for the cluster. Blocks already allocated for the table are reorganized when necessary (not immediately).

When you alter the transaction entry setting INITRANS of a cluster, the new setting for INITRANS applies only to data blocks subsequently allocated for the cluster.

The storage parameters INITIAL and MINEXTENTS cannot be altered. All new settings for the other storage parameters affect only extents subsequently allocated for the cluster.

To alter a cluster, use the ALTER CLUSTER statement. The following statement alters the emp_dept cluster:

```
ALTER CLUSTER emp_dept
   PCTFREE 30
   PCTUSED 60;
```

See Also: *Oracle Database SQL Reference* for syntax of the ALTER CLUSTER statement

Altering Clustered Tables

You can alter clustered tables using the ALTER TABLE statement. However, any data block space parameters, transaction entry parameters, or storage parameters you set in an ALTER TABLE statement for a clustered table generate an error message (`ORA-01771, illegal option for a clustered table`). The database uses the parameters of the cluster for all clustered tables. Therefore, you can use the ALTER TABLE statement only to add or modify columns, drop non-cluster-key columns, or add, drop, enable, or disable integrity constraints or triggers for a clustered table. For information about altering tables, see "Altering Tables" on page 14-18.

See Also: *Oracle Database SQL Reference* for syntax of the ALTER TABLE statement

Altering Cluster Indexes

You alter cluster indexes exactly as you do other indexes. See "Altering Indexes" on page 15-16.

Note: When estimating the size of cluster indexes, remember that the index is on each cluster key, not the actual rows. Therefore, each key appears only once in the index.

Dropping Clusters

A cluster can be dropped if the tables within the cluster are no longer needed. When a cluster is dropped, so are the tables within the cluster and the corresponding cluster index. All extents belonging to both the cluster data segment and the index segment of the cluster index are returned to the containing tablespace and become available for other segments within the tablespace.

To drop a cluster that contains no tables, and its cluster index, use the DROP CLUSTER statement. For example, the following statement drops the empty cluster named `emp_dept`:

```
DROP CLUSTER emp_dept;
```

If the cluster contains one or more clustered tables and you intend to drop the tables as well, add the INCLUDING TABLES clause of the DROP CLUSTER statement, as follows:

```
DROP CLUSTER emp_dept INCLUDING TABLES;
```

If the INCLUDING TABLES clause is not included and the cluster contains tables, an error is returned.

If one or more tables in a cluster contain primary or unique keys that are referenced by FOREIGN KEY constraints of tables outside the cluster, the cluster cannot be dropped unless the dependent FOREIGN KEY constraints are also dropped. This can be easily done using the CASCADE CONSTRAINTS clause of the DROP CLUSTER statement, as shown in the following example:

```
DROP CLUSTER emp_dept INCLUDING TABLES CASCADE CONSTRAINTS;
```

The database returns an error if you do not use the CASCADE CONSTRAINTS clause and constraints exist.

> **See Also:** *Oracle Database SQL Reference* for syntax of the DROP CLUSTER statement

Dropping Clustered Tables

To drop a cluster, your schema must contain the cluster or you must have the DROP ANY CLUSTER system privilege. You do not need additional privileges to drop a cluster that contains tables, even if the clustered tables are not owned by the owner of the cluster.

Clustered tables can be dropped individually without affecting the cluster, other clustered tables, or the cluster index. A clustered table is dropped just as a nonclustered table is dropped, with the DROP TABLE statement. See "Dropping Table Columns" on page 14-23.

Note: When you drop a single table from a cluster, the database deletes each row of the table individually. To maximize efficiency when you intend to drop an entire cluster, drop the cluster including all tables by using the DROP CLUSTER statement with the INCLUDING TABLES clause. Drop an individual table from a cluster (using the DROP TABLE statement) only if you want the rest of the cluster to remain.

Dropping Cluster Indexes

A cluster index can be dropped without affecting the cluster or its clustered tables. However, clustered tables cannot be used if there is no cluster index; you must re-create the cluster index to allow access to the cluster. Cluster indexes are sometimes dropped as part of the procedure to rebuild a fragmented cluster index. For information about dropping an index, see "Dropping Indexes" on page 15-20.

Viewing Information About Clusters

The following views display information about clusters:

View	Description
DBA_CLUSTERS ALL_CLUSTERS USER_CLUSTERS	DBA view describes all clusters in the database. ALL view describes all clusters accessible to the user. USER view is restricted to clusters owned by the user. Some columns in these views contain statistics that are generated by the DBMS_STATS package or ANALYZE statement.
DBA_CLU_COLUMNS USER_CLU_COLUMNS	These views map table columns to cluster columns

See Also: *Oracle Database Reference* for complete descriptions of these views

18

Managing Hash Clusters

This chapter describes how to manage hash clusters, and contains the following topics:

About Hash Clusters

Storing a table in a hash cluster is an optional way to improve the performance of data retrieval. A hash cluster provides an alternative to a nonclustered table with an index or an index cluster. With an indexed table or index cluster, Oracle Database locates the rows in a table using key values that the database stores in a separate index. To use hashing, you create a hash cluster and load tables into it. The database physically stores the rows of a table in a hash cluster and retrieves them according to the results of a **hash function**.

Oracle Database uses a hash function to generate a distribution of numeric values, called **hash values**, that are based on specific cluster key values. The key of a hash cluster, like the key of an index cluster, can be a single column or composite key (multiple column key). To find or store a row in a hash cluster, the database applies the hash function to the cluster key value of the row. The resulting hash value corresponds to a data block in the cluster, which the database then reads or writes on behalf of the issued statement.

To find or store a row in an indexed table or cluster, a minimum of two (there are usually more) I/Os must be performed:

- One or more I/Os to find or store the key value in the index
- Another I/O to read or write the row in the table or cluster

In contrast, the database uses a hash function to locate a row in a hash cluster; no I/O is required. As a result, a minimum of one I/O operation is necessary to read or write a row in a hash cluster.

> **See Also:** Chapter 13, "Managing Space for Schema Objects" is recommended reading before attempting tasks described in this chapter.

When to Use Hash Clusters

This section helps you decide when to use hash clusters by contrasting situations where hashing is most useful against situations where there is no advantage. If you find your decision is to use indexing rather than hashing, then you should consider whether to store a table individually or as part of a cluster.

> **Note:** Even if you decide to use hashing, a table can still have separate indexes on any columns, including the cluster key.

Situations Where Hashing Is Useful

Hashing is useful when you have the following conditions:

- Most queries are equality queries on the cluster key:

  ```
  SELECT ... WHERE cluster_key = ...;
  ```

  In such cases, the cluster key in the equality condition is hashed, and the corresponding hash key is usually found with a single read. In comparison, for an indexed table the key value must first be found in the index (usually several reads), and then the row is read from the table (another read).

- The tables in the hash cluster are primarily static in size so that you can determine the number of rows and amount of space required for the tables in the cluster. If tables in a hash cluster require more space than the initial allocation for the cluster, performance degradation can be substantial because overflow blocks are required.

Situations Where Hashing Is Not Advantageous

Hashing is not advantageous in the following situations:

- Most queries on the table retrieve rows over a range of cluster key values. For example, in full table scans or queries such as the following, a hash function cannot be used to determine the location of specific hash keys. Instead, the equivalent of a full table scan must be done to fetch the rows for the query.

  ```
  SELECT . . . WHERE cluster_key < . . . ;
  ```

  With an index, key values are ordered in the index, so cluster key values that satisfy the WHERE clause of a query can be found with relatively few I/Os.

- The table is not static, but instead is continually growing. If a table grows without limit, the space required over the life of the table (its cluster) cannot be predetermined.
- Applications frequently perform full-table scans on the table and the table is sparsely populated. A full-table scan in this situation takes longer under hashing.
- You cannot afford to preallocate the space that the hash cluster will eventually need.

Creating Hash Clusters

A hash cluster is created using a CREATE CLUSTER statement, but you specify a HASHKEYS clause. The following example contains a statement to create a cluster named `trial_cluster` that stores the `trial` table, clustered by the `trialno` column (the cluster key); and another statement creating a table in the cluster.

```
CREATE CLUSTER trial_cluster (trialno NUMBER(5,0))
    PCTUSED 80 PCTFREE 5
    TABLESPACE users
    STORAGE (INITIAL 250K     NEXT 50K
      MINEXTENTS 1     MAXEXTENTS 3
      PCTINCREASE 0)
    HASH IS trialno HASHKEYS 150;

CREATE TABLE trial (
    trialno NUMBER(5,0) PRIMARY KEY,
    ...)
    CLUSTER trial_cluster (trialno);
```

As with index clusters, the key of a hash cluster can be a single column or a composite key (multiple column key). In this example, it is a single column.

The HASHKEYS value, in this case 150, specifies and limits the number of unique hash values that can be generated by the hash function used by the cluster. The database rounds the number specified to the nearest prime number.

If no HASH IS clause is specified, the database uses an internal hash function. If the cluster key is already a unique identifier that is uniformly distributed over its range, you can bypass the internal hash function and specify the cluster key as the hash value, as is the case in the preceding example. You can also use the HASH IS clause to specify a user-defined hash function.

You cannot create a cluster index on a hash cluster, and you need not create an index on a hash cluster key.

For additional information about creating tables in a cluster, guidelines for setting parameters of the CREATE CLUSTER statement common to index and hash clusters, and the privileges required to create any cluster, see Chapter 17, "Managing Clusters". The following sections explain and provide guidelines for setting the parameters of the CREATE CLUSTER statement specific to hash clusters:

- Creating a Sorted Hash Cluster
- Creating Single-Table Hash Clusters
- Controlling Space Use Within a Hash Cluster
- Estimating Size Required by Hash Clusters

Creating a Sorted Hash Cluster

In a **sorted hash cluster**, the rows corresponding to each value of the hash function are sorted on a specified set of columns in ascending order, which can improve response time during subsequent operations on the clustered data.

For example, a telecommunications company needs to store detailed call records for a fixed number of originating telephone numbers through a telecommunications switch. From each originating telephone number there can be an unlimited number of telephone calls.

Calls are stored as they are made and processed later in first-in, first-out order (FIFO) when bills are generated for each originating telephone number. Each call has a detailed call record that is identified by a timestamp. The data that is gathered is similar to the following:

Originating Telephone Numbers	Call Records Identified by Timestamp
650-555-1212	t0, t1, t2, t3, t4, ...
650-555-1213	t0, t1, t2, t3, t4, ...
650-555-1214	t0, t1, t2, t3, t4, ...
...	...

In the following SQL statements, the `telephone_number` column is the hash key. The hash cluster is sorted on the `call_timestamp` and `call_duration` columns. The number of hash keys is based on 10-digit telephone numbers.

```
CREATE CLUSTER call_detail_cluster (
   telephone_number NUMBER,
   call_timestamp NUMBER SORT,
   call_duration NUMBER SORT )
  HASHKEYS 10000 HASH IS telephone_number
  SIZE 256;

CREATE TABLE call_detail (
   telephone_number     NUMBER,
   call_timestamp       NUMBER   SORT,
   call_duration        NUMBER   SORT,
   other_info           VARCHAR2(30) )
  CLUSTER call_detail_cluster (
   telephone_number, call_timestamp, call_duration );
```

Given the sort order of the data, the following query would return the call records for a specified hash key by oldest record first.

```
SELECT * WHERE telephone_number = 6505551212;
```

Creating Single-Table Hash Clusters

You can also create a **single-table hash cluster**, which provides fast access to rows in a table. However, this table must be the only table in the hash cluster. Essentially, there must be a one-to-one mapping between hash keys and data rows. The following statement creates a single-table hash cluster named `peanut` with the cluster key `variety`:

```
CREATE CLUSTER peanut (variety NUMBER)
   SIZE 512 SINGLE TABLE HASHKEYS 500;
```

The database rounds the HASHKEYS value up to the nearest prime number, so this cluster has a maximum of 503 hash key values, each of size 512 bytes. The SINGLE TABLE clause is valid only for hash clusters. HASHKEYS must also be specified.

> **See Also:** *Oracle Database SQL Reference* for the syntax of the CREATE CLUSTER statement

Controlling Space Use Within a Hash Cluster

When creating a hash cluster, it is important to choose the cluster key correctly and set the HASH IS, SIZE, and HASHKEYS parameters so that performance and space use are optimal. The following guidelines describe how to set these parameters.

Choosing the Key

Choosing the correct cluster key is dependent on the most common types of queries issued against the clustered tables. For example, consider the emp table in a hash cluster. If queries often select rows by employee number, the empno column should be the cluster key. If queries often select rows by department number, the deptno column should be the cluster key. For hash clusters that contain a single table, the cluster key is typically the entire primary key of the contained table.

The key of a hash cluster, like that of an index cluster, can be a single column or a composite key (multiple column key). A hash cluster with a composite key must use the internal hash function of the database.

Setting HASH IS

Specify the HASH IS parameter only if the cluster key is a single column of the NUMBER datatype, and contains uniformly distributed integers. If these conditions apply, you can distribute rows in the cluster so that each unique cluster key value hashes, with no collisions (two cluster key values having the same hash value), to a unique hash value. If these conditions do not apply, omit this clause so that you use the internal hash function.

Setting SIZE

SIZE should be set to the average amount of space required to hold all rows for any given hash key. Therefore, to properly determine SIZE, you must be aware of the characteristics of your data:

- If the hash cluster is to contain only a single table and the hash key values of the rows in that table are unique (one row for each value), SIZE can be set to the average row size in the cluster.

- If the hash cluster is to contain multiple tables, SIZE can be set to the average amount of space required to hold all rows associated with a representative hash value.

Further, once you have determined a (preliminary) value for SIZE, consider the following. If the SIZE value is small (more than four hash keys can be assigned for each data block) you can use this value for SIZE in the CREATE CLUSTER statement. However, if the value of SIZE is large (four or fewer hash keys can be assigned for each data block), then you should also consider the expected frequency of collisions and whether performance of data retrieval or efficiency of space usage is more important to you.

- If the hash cluster does not use the internal hash function (if you specified HASH IS) and you expect few or no collisions, you can use your preliminary value of SIZE. No collisions occur and space is used as efficiently as possible.
- If you expect frequent collisions on inserts, the likelihood of overflow blocks being allocated to store rows is high. To reduce the possibility of overflow blocks and maximize performance when collisions are frequent, you should adjust SIZE as shown in the following chart.

Available Space for each Block / Calculated SIZE	Setting for SIZE
1	SIZE
2	SIZE + 15%
3	SIZE + 12%
4	SIZE + 8%
>4	SIZE

Overestimating the value of SIZE increases the amount of unused space in the cluster. If space efficiency is more important than the performance of data retrieval, disregard the adjustments shown in the preceding table and use the original value for SIZE.

Setting HASHKEYS

For maximum distribution of rows in a hash cluster, the database rounds the HASHKEYS value up to the nearest prime number.

Controlling Space in Hash Clusters

The following examples show how to correctly choose the cluster key and set the HASH IS, SIZE, and HASHKEYS parameters. For all examples, assume that the data block size is 2K and that on average, 1950 bytes of each block is available data space (block size minus overhead).

Controlling Space in Hash Clusters: Example 1 You decide to load the emp table into a hash cluster. Most queries retrieve employee records by their employee number. You estimate that the maximum number of rows in the emp table at any given time is 10000 and that the average row size is 55 bytes.

In this case, empno should be the cluster key. Because this column contains integers that are unique, the internal hash function can be bypassed. SIZE can be set to the average row size, 55 bytes. Note that 34 hash keys are assigned for each data block. HASHKEYS can be set to the number of rows in the table, 10000. The database rounds this value up to the next highest prime number: 10007.

```
CREATE CLUSTER emp_cluster (empno
NUMBER)
. . .
SIZE 55
HASH IS empno HASHKEYS 10000;
```

Controlling Space in Hash Clusters: Example 2 Conditions similar to the previous example exist. In this case, however, rows are usually retrieved by department number. At most, there are 1000 departments with an average of 10 employees for each department. Department numbers increment by 10 (0, 10, 20, 30, . . .).

In this case, deptno should be the cluster key. Since this column contains integers that are uniformly distributed, the internal hash function can be bypassed. A preliminary value of SIZE (the average amount of space required to hold all rows for each department) is 55 bytes * 10, or 550 bytes. Using this value for SIZE, only three hash keys can be assigned for each data block. If you expect some collisions and want maximum performance of data retrieval, slightly alter your estimated SIZE to prevent collisions from requiring overflow blocks. By adjusting SIZE by 12%, to 620 bytes (refer to "Setting SIZE" on page 18-6), there is more space for rows from expected collisions.

HASHKEYS can be set to the number of unique department numbers, 1000. The database rounds this value up to the next highest prime number: 1009.

```
CREATE CLUSTER emp_cluster (deptno NUMBER)
. . .
SIZE 620
```

```
HASH IS deptno HASHKEYS 1000;
```

Estimating Size Required by Hash Clusters

As with index clusters, it is important to estimate the storage required for the data in a hash cluster.

Oracle Database guarantees that the initial allocation of space is sufficient to store the hash table according to the settings SIZE and HASHKEYS. If settings for the storage parameters INITIAL, NEXT, and MINEXTENTS do not account for the hash table size, incremental (additional) extents are allocated until at least SIZE*HASHKEYS is reached. For example, assume that the data block size is 2K, the available data space for each block is approximately 1900 bytes (data block size minus overhead), and that the STORAGE and HASH parameters are specified in the CREATE CLUSTER statement as follows:

```
STORAGE (INITIAL 100K
    NEXT 150K
    MINEXTENTS 1
    PCTINCREASE 0)
SIZE 1500
HASHKEYS 100
```

In this example, only one hash key can be assigned for each data block. Therefore, the initial space required for the hash cluster is at least 100*2K or 200K. The settings for the storage parameters do not account for this requirement. Therefore, an initial extent of 100K and a second extent of 150K are allocated to the hash cluster.

Alternatively, assume the HASH parameters are specified as follows:

```
SIZE 500 HASHKEYS 100
```

In this case, three hash keys are assigned to each data block. Therefore, the initial space required for the hash cluster is at least 34*2K or 68K. The initial settings for the storage parameters are sufficient for this requirement (an initial extent of 100K is allocated to the hash cluster).

Altering Hash Clusters

You can alter a hash cluster with the ALTER CLUSTER statement:

```
ALTER CLUSTER emp_dept . . . ;
```

The implications for altering a hash cluster are identical to those for altering an index cluster, described in "Altering Clusters" on page 17-9. However, the SIZE, HASHKEYS, and HASH IS parameters cannot be specified in an ALTER CLUSTER statement. To change these parameters, you must re-create the cluster, then copy the data from the original cluster.

Dropping Hash Clusters

You can drop a hash cluster using the DROP CLUSTER statement:

```
DROP CLUSTER emp_dept;
```

A table in a hash cluster is dropped using the DROP TABLE statement. The implications of dropping hash clusters and tables in hash clusters are the same as those for dropping index clusters.

See Also: "Dropping Clusters" on page 17-10

Viewing Information About Hash Clusters

The following views display information about hash clusters:

View	Description
DBA_CLUSTERS ALL_CLUSTERS USER_CLUSTERS	DBA view describes all clusters (including hash clusters) in the database. ALL view describes all clusters accessible to the user. USER view is restricted to clusters owned by the user. Some columns in these views contain statistics that are generated by the DBMS_STATS package or ANALYZE statement.
DBA_CLU_COLUMNS USER_CLU_COLUMNS	These views map table columns to cluster columns.
DBA_CLUSTER_HASH_EXPRESSIONS ALL_CLUSTER_HASH_EXPRESSIONS USER_CLUSTER_HASH_EXPRESSIONS	These views list hash functions for hash clusters.

See Also: *Oracle Database Reference* for complete descriptions of these views

19

Managing Views, Sequences, and Synonyms

This chapter describes the management of views, sequences, and synonyms and contains the following topics:

Managing Views

This section describes aspects of managing views, and contains the following topics:

About Views

A **view** is a logical representation of another table or combination of tables. A view derives its data from the tables on which it is based. These tables are called **base tables**. Base tables might in turn be actual tables or might be views themselves. All operations performed on a view actually affect the base table of the view. You can use views in almost the same way as tables. You can query, update, insert into, and delete from views, just as you can standard tables.

Views can provide a different representation (such as subsets or supersets) of the data that resides within other tables and views. Views are very powerful because they allow you to tailor the presentation of data to different types of users.

> **See Also:** *Oracle Database Concepts* for a more complete description of views

Creating Views

To create a view, you must meet the following requirements:

- To create a view in your schema, you must have the `CREATE VIEW` privilege. To create a view in another user's schema, you must have the `CREATE ANY VIEW` system privilege. You can acquire these privileges explicitly or through a role.
- The owner of the view (whether it is you or another user) must have been explicitly granted privileges to access all objects referenced in the view definition. The owner *cannot* have obtained these privileges through roles. Also, the functionality of the view is dependent on the privileges of the view owner. For example, if the owner of the view has only the `INSERT` privilege for Scott's `emp` table, the view can only be used to insert new rows into the `emp` table, not to `SELECT`, `UPDATE`, or `DELETE` rows.
- If the owner of the view intends to grant access to the view to other users, the owner must have received the object privileges to the base objects with the `GRANT OPTION` or the system privileges with the `ADMIN OPTION`.

You can create views using the `CREATE VIEW` statement. Each view is defined by a query that references tables, materialized views, or other views. As with all subqueries, the query that defines a view cannot contain the `FOR UPDATE` clause.

The following statement creates a view on a subset of data in the `emp` table:

```
CREATE VIEW sales_staff AS
      SELECT empno, ename, deptno
      FROM emp
```

```
    WHERE deptno = 10
WITH CHECK OPTION CONSTRAINT sales_staff_cnst;
```

The query that defines the sales_staff view references only rows in department 10. Furthermore, the CHECK OPTION creates the view with the constraint (named sales_staff_cnst) that INSERT and UPDATE statements issued against the view cannot result in rows that the view cannot select. For example, the following INSERT statement successfully inserts a row into the emp table by means of the sales_staff view, which contains all rows with department number 10:

```
INSERT INTO sales_staff VALUES (7584, 'OSTER', 10);
```

However, the following INSERT statement returns an error because it attempts to insert a row for department number 30, which cannot be selected using the sales_staff view:

```
INSERT INTO sales_staff VALUES (7591, 'WILLIAMS', 30);
```

The view could optionally have been constructed specifying the WITH READ ONLY clause, which prevents any updates, inserts, or deletes from being done to the base table through the view. If no WITH clause is specified, the view, with some restrictions, is inherently updatable.

> **See Also:** *Oracle Database SQL Reference* for syntax and semantics of the CREATE VIEW statement

Join Views

You can also create views that specify more than one base table or view in the FROM clause. These are called **join views**. The following statement creates the division1_staff view that joins data from the emp and dept tables:

```
CREATE VIEW division1_staff AS
      SELECT ename, empno, job, dname
      FROM emp, dept
      WHERE emp.deptno IN (10, 30)
         AND emp.deptno = dept.deptno;
```

An **updatable join view** is a join view where UPDATE, INSERT, and DELETE operations are allowed. See "Updating a Join View" on page 19-7 for further discussion.

Expansion of Defining Queries at View Creation Time

When a view is created, Oracle Database expands any wildcard (*) in a top-level view query into a column list. The resulting query is stored in the data dictionary; any subqueries are left intact. The column names in an expanded column list are enclosed in quote marks to account for the possibility that the columns of the base object were originally entered with quotes and require them for the query to be syntactically correct.

As an example, assume that the `dept` view is created as follows:

```
CREATE VIEW dept AS SELECT * FROM scott.dept;
```

The database stores the defining query of the `dept` view as:

```
SELECT "DEPTNO", "DNAME", "LOC" FROM scott.dept;
```

Views created with errors do not have wildcards expanded. However, if the view is eventually compiled without errors, wildcards in the defining query are expanded.

Creating Views with Errors

If there are no syntax errors in a `CREATE VIEW` statement, the database can create the view even if the defining query of the view cannot be executed. In this case, the view is considered "created with errors." For example, when a view is created that refers to a nonexistent table or an invalid column of an existing table, or when the view owner does not have the required privileges, the view can be created anyway and entered into the data dictionary. However, the view is not yet usable.

To create a view with errors, you must include the `FORCE` clause of the `CREATE VIEW` statement.

```
CREATE FORCE VIEW AS ...;
```

By default, views with errors are created as `INVALID`. When you try to create such a view, the database returns a message indicating the view was created with errors. If conditions later change so that the query of an invalid view can be executed, the view can be recompiled and be made valid (usable). For information changing conditions and their impact on views, see "Managing Object Dependencies" on page 20-22.

Replacing Views

To replace a view, you must have all the privileges required to drop and create a view. If the definition of a view must change, the view must be replaced; you cannot

use an ALTER VIEW statement to change the definition of a view. You can replace views in the following ways:

- You can drop and re-create the view.

  Caution: When a view is dropped, all grants of corresponding object privileges are revoked from roles and users. After the view is re-created, privileges must be regranted.

- You can redefine the view with a CREATE VIEW statement that contains the OR REPLACE clause. The OR REPLACE clause replaces the current definition of a view and preserves the current security authorizations. For example, assume that you created the sales_staff view as shown earlier, and, in addition, you granted several object privileges to roles and other users. However, now you need to redefine the sales_staff view to change the department number specified in the WHERE clause. You can replace the current version of the sales_staff view with the following statement:

```
CREATE OR REPLACE VIEW sales_staff AS
     SELECT empno, ename, deptno
     FROM emp
     WHERE deptno = 30
     WITH CHECK OPTION CONSTRAINT sales_staff_cnst;
```

Before replacing a view, consider the following effects:

- Replacing a view replaces the view definition in the data dictionary. All underlying objects referenced by the view are not affected.
- If a constraint in the CHECK OPTION was previously defined but not included in the new view definition, the constraint is dropped.
- All views and PL/SQL program units dependent on a replaced view become invalid (not usable). See "Managing Object Dependencies" on page 20-22 for more information on how the database manages such dependencies.

Using Views in Queries

To issue a query or an INSERT, UPDATE, or DELETE statement against a view, you must have the SELECT, INSERT, UPDATE, or DELETE object privilege for the view, respectively, either explicitly or through a role.

Views can be queried in the same manner as tables. For example, to query the `Division1_staff` view, enter a valid `SELECT` statement that references the view:

```
SELECT * FROM Division1_staff;

ENAME        EMPNO       JOB             DNAME
---------------------------------------------------
CLARK         7782       MANAGER         ACCOUNTING
KING          7839       PRESIDENT       ACCOUNTING
MILLER        7934       CLERK           ACCOUNTING
ALLEN         7499       SALESMAN        SALES
WARD          7521       SALESMAN        SALES
JAMES         7900       CLERK           SALES
TURNER        7844       SALESMAN        SALES
MARTIN        7654       SALESMAN        SALES
BLAKE         7698       MANAGER         SALES
```

With some restrictions, rows can be inserted into, updated in, or deleted from a base table using a view. The following statement inserts a new row into the `emp_tab` table using the `sales_staff` view:

```
INSERT INTO Sales_staff
    VALUES (7954, 'OSTER', 30);
```

Restrictions on DML operations for views use the following criteria in the order listed:

1. If a view is defined by a query that contains `SET` or `DISTINCT` operators, a `GROUP BY` clause, or a group function, then rows cannot be inserted into, updated in, or deleted from the base tables using the view.
2. If a view is defined with `WITH CHECK OPTION`, then a row cannot be inserted into, or updated in, the base table (using the view), if the view cannot select the row from the base table.
3. If a `NOT NULL` column that does not have a `DEFAULT` clause is omitted from the view, then a row cannot be inserted into the base table using the view.
4. If the view was created by using an expression, such as `DECODE`(deptno, 10, "SALES", ...), then rows cannot be inserted into or updated in the base table using the view.

The constraint created by `WITH CHECK OPTION` of the `sales_staff` view only allows rows that have a department number of 10 to be inserted into, or updated in, the `emp_tab` table. Alternatively, assume that the `sales_staff` view is defined by the following statement (that is, excluding the `deptno` column):

```
CREATE VIEW Sales_staff AS
    SELECT Empno, Ename
    FROM Emp_tab
    WHERE Deptno = 10
    WITH CHECK OPTION CONSTRAINT Sales_staff_cnst;
```

Considering this view definition, you can update the empno or ename fields of existing records, but you cannot insert rows into the emp_tab table through the sales_staff view because the view does not let you alter the deptno field. If you had defined a DEFAULT value of 10 on the deptno field, then you could perform inserts.

When a user attempts to reference an invalid view, the database returns an error message to the user:

```
ORA-04063: view 'view_name' has errors
```

This error message is returned when a view exists but is unusable due to errors in its query (whether it had errors when originally created or it was created successfully but became unusable later because underlying objects were altered or dropped).

Updating a Join View

An updatable join view (also referred to as a **modifiable join view**) is a view that contains more than one table in the top-level FROM clause of the SELECT statement, and is not restricted by the WITH READ ONLY clause.

The rules for updatable join views are as follows. Views that meet this criteria are said to be inherently updatable.

Rule	Description
General Rule	Any INSERT, UPDATE, or DELETE operation on a join view can modify only one underlying base table at a time.
UPDATE Rule	All updatable columns of a join view must map to columns of a **key-preserved table**. See "Key-Preserved Tables" on page 19-9 for a discussion of key-preserved tables. If the view is defined with the WITH CHECK OPTION clause, then all join columns and all columns of repeated tables are not updatable.
DELETE Rule	Rows from a join view can be deleted as long as there is exactly one key-preserved table in the join. If the view is defined with the WITH CHECK OPTION clause and the key preserved table is repeated, then the rows cannot be deleted from the view.

Rule	Description
INSERT Rule	An INSERT statement must not explicitly or implicitly refer to the columns of a **non-key-preserved table**. If the join view is defined with the WITH CHECK OPTION clause, INSERT statements are not permitted.

There are data dictionary views that indicate whether the columns in a join view are inherently updatable. See "Using the UPDATABLE_ COLUMNS Views" on page 19-15 for descriptions of these views.

Note: There are some additional restrictions and conditions which can affect whether a join view is inherently updatable. Specifics are listed in the description of the CREATE VIEW statement in the *Oracle Database SQL Reference.*

If a view is not inherently updatable, it can be made updatable by creating an INSTEAD OF trigger on it. This is described in *Oracle Database Application Developer's Guide - Fundamentals.*

Additionally, if a view is a join on other nested views, then the other nested views must be mergeable into the top level view. For a discussion of mergeable and unmergeable views, and more generally, how the optimizer optimizes statements referencing views, see the *Oracle Database Performance Tuning Guide.*

Examples illustrating the rules for inherently updatable join views, and a discussion of key-preserved tables, are presented in succeeding sections. The examples in these sections work only if you explicitly define the primary and foreign keys in the tables, or define unique indexes. The following statements create the appropriately constrained table definitions for emp and dept.

```
CREATE TABLE dept (
      deptno        NUMBER(4) PRIMARY KEY,
      dname         VARCHAR2(14),
      loc           VARCHAR2(13));

CREATE TABLE emp (
      empno        NUMBER(4) PRIMARY KEY,
      ename        VARCHAR2(10),
      job          VARCHAR2(9),
      mgr          NUMBER(4),
      sal          NUMBER(7,2),
```

```
      comm         NUMBER(7,2),
      deptno       NUMBER(2),
      FOREIGN KEY (DEPTNO) REFERENCES DEPT(DEPTNO));
```

You could also omit the primary and foreign key constraints listed in the preceding example, and create a `UNIQUE INDEX` on `dept (deptno)` to make the following examples work.

The following statement created the `emp_dept` join view which is referenced in the examples:

```
CREATE VIEW emp_dept AS
      SELECT emp.empno, emp.ename, emp.deptno, emp.sal, dept.dname, dept.loc
      FROM emp, dept
      WHERE emp.deptno = dept.deptno
         AND dept.loc IN ('DALLAS', 'NEW YORK', 'BOSTON');
```

Key-Preserved Tables

The concept of a **key-preserved table** is fundamental to understanding the restrictions on modifying join views. A table is key preserved if every key of the table can also be a key of the result of the join. So, a key-preserved table has its keys preserved through a join.

> **Note:** It is not necessary that the key or keys of a table be selected for it to be key preserved. It is sufficient that if the key or keys were selected, then they would also be key(s) of the result of the join.

The key-preserving property of a table does not depend on the actual data in the table. It is, rather, a property of its schema. For example, if in the `emp` table there was at most one employee in each department, then `deptno` would be unique in the result of a join of `emp` and `dept`, but `dept` would still not be a key-preserved table.

If you `SELECT` all rows from `emp_dept`, the results are:

```
EMPNO      ENAME      DEPTNO  DNAME          LOC
---------- ---------- ------- -------------- -----------
      7782 CLARK           10 ACCOUNTING     NEW YORK
      7839 KING            10 ACCOUNTING     NEW YORK
      7934 MILLER          10 ACCOUNTING     NEW YORK
      7369 SMITH           20 RESEARCH       DALLAS
      7876 ADAMS           20 RESEARCH       DALLAS
      7902 FORD            20 RESEARCH       DALLAS
```

```
      7788 SCOTT           20 RESEARCH       DALLAS
      7566 JONES           20 RESEARCH       DALLAS
8 rows selected.
```

In this view, emp is a key-preserved table, because empno is a key of the emp table, and also a key of the result of the join. dept is *not* a key-preserved table, because although deptno is a key of the dept table, it is not a key of the join.

DML Statements and Join Views

The general rule is that any UPDATE, DELETE, or INSERT statement on a join view can modify only one underlying base table. The following examples illustrate rules specific to UPDATE, DELETE, and INSERT statements.

UPDATE Statements The following example shows an UPDATE statement that successfully modifies the emp_dept view:

```
UPDATE emp_dept
     SET sal = sal * 1.10
     WHERE deptno = 10;
```

The following UPDATE statement would be disallowed on the emp_dept view:

```
UPDATE emp_dept
     SET loc = 'BOSTON'
     WHERE ename = 'SMITH';
```

This statement fails with an error (ORA-01779 cannot modify a column which maps to a non key-preserved table), because it attempts to modify the base dept table, and the dept table is not key preserved in the emp_dept view.

In general, all updatable columns of a join view must map to columns of a key-preserved table. If the view is defined using the WITH CHECK OPTION clause, then all join columns and all columns taken from tables that are referenced more than once in the view are not modifiable.

So, for example, if the emp_dept view were defined using WITH CHECK OPTION, the following UPDATE statement would fail:

```
UPDATE emp_dept
     SET deptno = 10
     WHERE ename = 'SMITH';
```

The statement fails because it is trying to update a join column.

> **See Also:** *Oracle Database SQL Reference* for syntax and additional information about the UPDATE statement

DELETE Statements You can delete from a join view provided there is *one and only one* key-preserved table in the join.

The following DELETE statement works on the emp_dept view:

```
DELETE FROM emp_dept
     WHERE ename = 'SMITH';
```

This DELETE statement on the emp_dept view is legal because it can be translated to a DELETE operation on the base emp table, and because the emp table is the only key-preserved table in the join.

If you were to create the following view, a DELETE operation could not be performed on the view because both e1 and e2 are key-preserved tables:

```
CREATE VIEW emp_emp AS
     SELECT e1.ename, e2.empno, deptno
     FROM emp e1, emp e2
     WHERE e1.empno = e2.empno;
```

If a view is defined using the WITH CHECK OPTION clause and the key-preserved table is repeated, then rows cannot be deleted from such a view.

```
CREATE VIEW emp_mgr AS
     SELECT e1.ename, e2.ename mname
     FROM emp e1, emp e2
     WHERE e1.mgr = e2.empno
     WITH CHECK OPTION;
```

No deletion can be performed on this view because the view involves a self-join of the table that is key preserved.

> **See Also:** *Oracle Database SQL Reference* for syntax and additional information about the DELETE statement

INSERT Statements The following INSERT statement on the emp_dept view succeeds:

```
INSERT INTO emp_dept (ename, empno, deptno)
   VALUES ('KURODA', 9010, 40);
```

This statement works because only one key-preserved base table is being modified (emp), and 40 is a valid deptno in the dept table (thus satisfying the FOREIGN KEY integrity constraint on the emp table).

An INSERT statement, such as the following, would fail for the same reason that such an UPDATE on the base emp table would fail: the FOREIGN KEY integrity constraint on the emp table is violated (because there is no deptno 77).

```
INSERT INTO emp_dept (ename, empno, deptno)
   VALUES ('KURODA', 9010, 77);
```

The following INSERT statement would fail with an error (ORA-01776 cannot modify more than one base table through a view):

```
INSERT INTO emp_dept (empno, ename, loc)
   VALUES (9010, 'KURODA', 'BOSTON');
```

An INSERT cannot implicitly or explicitly refer to columns of a non-key-preserved table. If the join view is defined using the WITH CHECK OPTION clause, then you cannot perform an INSERT to it.

> **See Also:** *Oracle Database SQL Reference* for syntax and additional information about the INSERT statement

Updating Views That Involve Outer Joins

Views that involve outer joins are modifiable in some cases. For example:

```
CREATE VIEW Emp_dept_oj1 AS
    SELECT Empno, Ename, e.Deptno, Dname, Loc
    FROM Emp_tab e, Dept_tab d
    WHERE e.Deptno = d.Deptno (+);
```

The statement:

```
SELECT * FROM Emp_dept_oj1;
```

Results in:

```
EMPNO   ENAME      DEPTNO  DNAME          LOC
------- ---------- ------- -------------- -------------
7369    SMITH      40      OPERATIONS     BOSTON
7499    ALLEN      30      SALES          CHICAGO
7566    JONES      20      RESEARCH       DALLAS
7654    MARTIN     30      SALES          CHICAGO
7698    BLAKE      30      SALES          CHICAGO
7782    CLARK      10      ACCOUNTING     NEW YORK
```

```
7788    SCOTT      20      RESEARCH       DALLAS
7839    KING       10      ACCOUNTING     NEW YORK
7844    TURNER     30      SALES          CHICAGO
7876    ADAMS      20      RESEARCH       DALLAS
7900    JAMES      30      SALES          CHICAGO
7902    FORD       20      RESEARCH       DALLAS
7934    MILLER     10      ACCOUNTING     NEW YORK
7521    WARD       30      SALES          CHICAGO
14 rows selected.
```

Columns in the base `emp_tab` table of `emp_dept_oj1` are modifiable through the view, because `emp_tab` is a key-preserved table in the join.

The following view also contains an outer join:

```
CREATE VIEW Emp_dept_oj2 AS
SELECT e.Empno, e.Ename, e.Deptno, d.Dname, d.Loc
FROM Emp_tab e, Dept_tab d
WHERE e.Deptno (+) = d.Deptno;
```

The statement:

```
SELECT * FROM Emp_dept_oj2;
```

Results in:

```
EMPNO      ENAME      DEPTNO    DNAME          LOC
---------- ---------- --------- -------------- ----
7782       CLARK      10        ACCOUNTING     NEW YORK
7839       KING       10        ACCOUNTING     NEW YORK
7934       MILLER     10        ACCOUNTING     NEW YORK
7369       SMITH      20        RESEARCH       DALLAS
7876       ADAMS      20        RESEARCH       DALLAS
7902       FORD       20        RESEARCH       DALLAS
7788       SCOTT      20        RESEARCH       DALLAS
7566       JONES      20        RESEARCH       DALLAS
7499       ALLEN      30        SALES          CHICAGO
7698       BLAKE      30        SALES          CHICAGO
7654       MARTIN     30        SALES          CHICAGO
7900       JAMES      30        SALES          CHICAGO
7844       TURNER     30        SALES          CHICAGO
7521       WARD       30        SALES          CHICAGO
                                 OPERATIONS     BOSTON
15 rows selected.
```

In this view, emp_tab is no longer a key-preserved table, because the empno column in the result of the join can have nulls (the last row in the preceding SELECT statement). So, UPDATE, DELETE, and INSERT operations cannot be performed on this view.

In the case of views containing an outer join on other nested views, a table is key preserved if the view or views containing the table are merged into their outer views, all the way to the top. A view which is being outer-joined is currently merged only if it is "simple." For example:

```
SELECT Col1, Col2, ... FROM T;
```

The select list of the view has no expressions, and there is no WHERE clause.

Consider the following set of views:

```
CREATE VIEW Emp_v AS
    SELECT Empno, Ename, Deptno
        FROM Emp_tab;
CREATE VIEW Emp_dept_oj1 AS
    SELECT e.*, Loc, d.Dname
        FROM Emp_v e, Dept_tab d
            WHERE e.Deptno = d.Deptno (+);
```

In these examples, emp_v is merged into emp_dept_oj1 because emp_v is a simple view, and so emp_tab is a key-preserved table. But if emp_v is changed as follows:

```
CREATE VIEW Emp_v_2 AS
    SELECT Empno, Ename, Deptno
        FROM Emp_tab
            WHERE Sal > 1000;
```

Then, because of the presence of the WHERE clause, emp_v_2 cannot be merged into emp_dept_oj1, and hence emp_tab is no longer a key-preserved table.

If you are in doubt whether a view is modifiable, then you can SELECT from the view USER_UPDATABLE_COLUMNS to see if it is. For example:

```
SELECT * FROM USER_UPDATABLE_COLUMNS WHERE TABLE_NAME = 'EMP_DEPT_VIEW';
```

This might return:

```
OWNER       TABLE_NAME      COLUMN_NAM      UPD
----------  ----------      ----------      ---
SCOTT       EMP_DEPT_V      EMPNO           NO
SCOTT       EMP_DEPT_V      ENAME           NO
```

```
SCOTT       EMP_DEPT_V      DEPTNO          NO
SCOTT       EMP_DEPT_V      DNAME           NO
SCOTT       EMP_DEPT_V      LOC             NO
5 rows selected.
```

Using the UPDATABLE_ COLUMNS Views

The views described in the following table can assist you to identify inherently updatable join views.

View	Description
DBA_UPDATABLE_ COLUMNS	Shows all columns in all tables and views that are modifiable.
ALL_UPDATABLE_ COLUMNS	Shows all columns in all tables and views accessible to the user that are modifiable.
USER_UPDATABLE_ COLUMNS	Shows all columns in all tables and views in the user's schema that are modifiable.

The updatable columns in view `emp_dept` are shown below.

```
SELECT COLUMN_NAME, UPDATABLE
      FROM USER_UPDATABLE_COLUMNS
      WHERE TABLE_NAME = 'EMP_DEPT';

COLUMN_NAME                    UPD
------------------------------ ---
EMPNO                          YES
ENAME                          YES
DEPTNO                         YES
SAL                            YES
DNAME                          NO
LOC                            NO

6 rows selected.
```

> **See Also:** *Oracle Database Reference* for complete descriptions of the updatable column views

Altering Views

You use the ALTER VIEW statement only to explicitly recompile a view that is invalid. If you want to change the definition of a view, see "Replacing Views" on page 19-4.

The ALTER VIEW statement lets you locate recompilation errors before run time. To ensure that the alteration does not affect the view or other objects that depend on it, you can explicitly recompile a view after altering one of its base tables.

To use the ALTER VIEW statement, the view must be in your schema, or you must have the ALTER ANY TABLE system privilege.

> **See Also:** *Oracle Database SQL Reference* for syntax and additional information about the ALTER VIEW statement

Dropping Views

You can drop any view contained in your schema. To drop a view in another user's schema, you must have the DROP ANY VIEW system privilege. Drop a view using the DROP VIEW statement. For example, the following statement drops the emp_dept view:

```
DROP VIEW emp_dept;
```

> **See Also:** *Oracle Database SQL Reference* for syntax and additional information about the DROP VIEW statement

Managing Sequences

This section describes aspects of managing sequences, and contains the following topics:

- About Sequences
- Creating Sequences
- Altering Sequences
- Using Sequences
- Dropping Sequences

About Sequences

Sequences are database objects from which multiple users can generate unique integers. The sequence generator generates sequential numbers, which can help to generate unique primary keys automatically, and to coordinate keys across multiple rows or tables.

Without sequences, sequential values can only be produced programmatically. A new primary key value can be obtained by selecting the most recently produced value and incrementing it. This method requires a lock during the transaction and causes multiple users to wait for the next value of the primary key; this waiting is known as **serialization**. If developers have such constructs in applications, then you should encourage the developers to replace them with access to sequences. Sequences eliminate serialization and improve the concurrency of an application.

> **See Also:** *Oracle Database Concepts* for a more complete description of sequences

Creating Sequences

To create a sequence in your schema, you must have the CREATE SEQUENCE system privilege. To create a sequence in another user's schema, you must have the CREATE ANY SEQUENCE privilege.

Create a sequence using the CREATE SEQUENCE statement. For example, the following statement creates a sequence used to generate employee numbers for the empno column of the emp table:

```
CREATE SEQUENCE emp_sequence
      INCREMENT BY 1
      START WITH 1
      NOMAXVALUE
      NOCYCLE
      CACHE 10;
```

Notice that several parameters can be specified to control the function of sequences. You can use these parameters to indicate whether the sequence is ascending or descending, the starting point of the sequence, the minimum and maximum values, and the interval between sequence values. The NOCYCLE option indicates that the sequence cannot generate more values after reaching its maximum or minimum value.

The CACHE clause preallocates a set of sequence numbers and keeps them in memory so that sequence numbers can be accessed faster. When the last of the

sequence numbers in the cache has been used, the database reads another set of numbers into the cache.

The database might skip sequence numbers if you choose to cache a set of sequence numbers. For example, when an instance abnormally shuts down (for example, when an instance failure occurs or a SHUTDOWN ABORT statement is issued), sequence numbers that have been cached but not used are lost. Also, sequence numbers that have been used but not saved are lost as well. The database might also skip cached sequence numbers after an export and import. See *Oracle Database Utilities* for details.

See Also:

- *Oracle Database SQL Reference* for the CREATE SEQUENCE statement syntax
- *Oracle Real Application Clusters Deployment and Performance Guide* for information about how caching sequence numbers improves performance in an Oracle Real Application Clusters environment

Altering Sequences

To alter a sequence, your schema must contain the sequence, or you must have the ALTER ANY SEQUENCE system privilege. You can alter a sequence to change any of the parameters that define how it generates sequence numbers except the sequence starting number. To change the starting point of a sequence, drop the sequence and then re-create it.

Alter a sequence using the ALTER SEQUENCE statement. For example, the following statement alters the emp_sequence:

```
ALTER SEQUENCE emp_sequence
    INCREMENT BY 10
    MAXVALUE 10000
    CYCLE
    CACHE 20;
```

See Also: *Oracle Database SQL Reference* for syntax and additional information about the ALTER SEQUENCE statement

Using Sequences

To use a sequence, your schema must contain the sequence or you must have been granted the SELECT object privilege for another user's sequence. Once a sequence is

defined, it can be accessed and incremented by multiple users (who have SELECT object privilege for the schema containing the sequence) with no waiting. The database does not wait for a transaction that has incremented a sequence to complete before that sequence can be incremented again.

The examples outlined in the following sections show how sequences can be used in master/detail table relationships. Assume an order entry system is partially comprised of two tables, `orders_tab` (master table) and `line_items_tab` (detail table), that hold information about customer orders. A sequence named `order_seq` is defined by the following statement:

```
CREATE SEQUENCE Order_seq
    START WITH 1
    INCREMENT BY 1
    NOMAXVALUE
    NOCYCLE
    CACHE 20;
```

Referencing a Sequence

A sequence is referenced in SQL statements with the NEXTVAL and CURRVAL pseudocolumns; each new sequence number is generated by a reference to the sequence pseudocolumn NEXTVAL, while the current sequence number can be repeatedly referenced using the pseudo-column CURRVAL.

NEXTVAL and CURRVAL are not reserved words or keywords and can be used as pseudocolumn names in SQL statements such as SELECT, INSERT, or UPDATE.

Generating Sequence Numbers with NEXTVAL To generate and use a sequence number, reference *seq_name*.NEXTVAL. For example, assume a customer places an order. The sequence number can be referenced in a values list. For example:

```
INSERT INTO Orders_tab (Orderno, Custno)
    VALUES (Order_seq.NEXTVAL, 1032);
```

Or, the sequence number can be referenced in the SET clause of an UPDATE statement. For example:

```
UPDATE Orders_tab
    SET Orderno = Order_seq.NEXTVAL
    WHERE Orderno = 10112;
```

The sequence number can also be referenced outermost SELECT of a query or subquery. For example:

```
SELECT Order_seq.NEXTVAL FROM dual;
```

As defined, the first reference to `order_seq.NEXTVAL` returns the value 1. Each subsequent statement that references `order_seq.NEXTVAL` generates the next sequence number (2, 3, 4,. . .). The pseudo-column NEXTVAL can be used to generate as many new sequence numbers as necessary. However, only a single sequence number can be generated for each row. In other words, if NEXTVAL is referenced more than once in a single statement, then the first reference generates the next number, and all subsequent references in the statement return the same number.

Once a sequence number is generated, the sequence number is available only to the session that generated the number. Independent of transactions committing or rolling back, other users referencing `order_seq.NEXTVAL` obtain unique values. If two users are accessing the same sequence concurrently, then the sequence numbers each user receives might have gaps because sequence numbers are also being generated by the other user.

Using Sequence Numbers with CURRVAL To use or refer to the current sequence value of your session, reference *seq_name*.CURRVAL. CURRVAL can only be used if *seq_name*.NEXTVAL has been referenced in the current user session (in the current or a previous transaction). CURRVAL can be referenced as many times as necessary, including multiple times within the same statement. The next sequence number is not generated until NEXTVAL is referenced. Continuing with the previous example, you would finish placing the customer's order by inserting the line items for the order:

```
INSERT INTO Line_items_tab (Orderno, Partno, Quantity)
    VALUES (Order_seq.CURRVAL, 20321, 3);

INSERT INTO Line_items_tab (Orderno, Partno, Quantity)
    VALUES (Order_seq.CURRVAL, 29374, 1);
```

Assuming the INSERT statement given in the previous section generated a new sequence number of 347, both rows inserted by the statements in this section insert rows with order numbers of 347.

Uses and Restrictions of NEXTVAL and CURRVAL CURRVAL and NEXTVAL can be used in the following places:

- VALUES clause of INSERT statements
- The SELECT list of a SELECT statement
- The SET clause of an UPDATE statement

CURRVAL and NEXTVAL cannot be used in these places:

- A subquery
- A view query or materialized view query
- A SELECT statement with the DISTINCT operator
- A SELECT statement with a GROUP BY or ORDER BY clause
- A SELECT statement that is combined with another SELECT statement with the UNION, INTERSECT, or MINUS set operator
- The WHERE clause of a SELECT statement
- DEFAULT value of a column in a CREATE TABLE or ALTER TABLE statement
- The condition of a CHECK constraint

Caching Sequence Numbers

Sequence numbers can be kept in the sequence cache in the System Global Area (SGA). Sequence numbers can be accessed more quickly in the sequence cache than they can be read from disk.

The sequence cache consists of entries. Each entry can hold many sequence numbers for a single sequence.

Follow these guidelines for fast access to all sequence numbers:

- Be sure the sequence cache can hold all the sequences used concurrently by your applications.
- Increase the number of values for each sequence held in the sequence cache.

The Number of Entries in the Sequence Cache When an application accesses a sequence in the sequence cache, the sequence numbers are read quickly. However, if an application accesses a sequence that is not in the cache, then the sequence must be read from disk to the cache before the sequence numbers are used.

If your applications use many sequences concurrently, then your sequence cache might not be large enough to hold all the sequences. In this case, access to sequence numbers might often require disk reads. For fast access to all sequences, be sure your cache has enough entries to hold all the sequences used concurrently by your applications.

The Number of Values in Each Sequence Cache Entry When a sequence is read into the sequence cache, sequence values are generated and stored in a cache entry. These values can then be accessed quickly. The number of sequence values stored in the

cache is determined by the CACHE parameter in the CREATE SEQUENCE statement. The default value for this parameter is 20.

This CREATE SEQUENCE statement creates the seq2 sequence so that 50 values of the sequence are stored in the SEQUENCE cache:

```
CREATE SEQUENCE Seq2
    CACHE 50;
```

The first 50 values of seq2 can then be read from the cache. When the 51st value is accessed, the next 50 values will be read from disk.

Choosing a high value for CACHE lets you access more successive sequence numbers with fewer reads from disk to the sequence cache. However, if there is an instance failure, then all sequence values in the cache are lost. Cached sequence numbers also could be skipped after an export and import if transactions continue to access the sequence numbers while the export is running.

If you use the NOCACHE option in the CREATE SEQUENCE statement, then the values of the sequence are not stored in the sequence cache. In this case, every access to the sequence requires a disk read. Such disk reads slow access to the sequence. This CREATE SEQUENCE statement creates the SEQ3 sequence so that its values are never stored in the cache:

```
CREATE SEQUENCE Seq3
    NOCACHE;
```

Dropping Sequences

You can drop any sequence in your schema. To drop a sequence in another schema, you must have the DROP ANY SEQUENCE system privilege. If a sequence is no longer required, you can drop the sequence using the DROP SEQUENCE statement. For example, the following statement drops the order_seq sequence:

```
DROP SEQUENCE order_seq;
```

When a sequence is dropped, its definition is removed from the data dictionary. Any synonyms for the sequence remain, but return an error when referenced.

> **See Also:** *Oracle Database SQL Reference* for syntax and additional information about the DROP SEQUENCE statement

Managing Synonyms

This section describes aspects of managing synonyms, and contains the following topics:

About Synonyms

A synonym is an alias for a schema object. Synonyms can provide a level of security by masking the name and owner of an object and by providing location transparency for remote objects of a distributed database. Also, they are convenient to use and reduce the complexity of SQL statements for database users.

Synonyms allow underlying objects to be renamed or moved, where only the synonym needs to be redefined and applications based on the synonym continue to function without modification.

You can create both public and private synonyms. A **public** synonym is owned by the special user group named `PUBLIC` and is accessible to every user in a database. A **private** synonym is contained in the schema of a specific user and available only to the user and the user's grantees.

> **See Also:** *Oracle Database Concepts* for a more complete description of synonyms

Creating Synonyms

To create a private synonym in your own schema, you must have the `CREATE SYNONYM` privilege. To create a private synonym in another user's schema, you must have the `CREATE ANY SYNONYM` privilege. To create a public synonym, you must have the `CREATE PUBLIC SYNONYM` system privilege.

Create a synonym using the `CREATE SYNONYM` statement. The underlying schema object need not exist, nor do you need privileges to access the object. The following statement creates a public synonym named `public_emp` on the `emp` table contained in the schema of `jward`:

```
CREATE PUBLIC SYNONYM public_emp FOR jward.emp
```

When you create a synonym for a remote procedure or function, you must qualify the remote object with its schema name. Alternatively, you can create a local public synonym on the database where the remote object resides, in which case the database link must be included in all subsequent calls to the procedure or function.

> **See Also:** *Oracle Database SQL Reference* for syntax and additional information about the CREATE SYNONYM statement

Using Synonyms in DML Statements

You can successfully use any private synonym contained in your schema or any public synonym, assuming that you have the necessary privileges to access the underlying object, either explicitly, from an enabled role, or from PUBLIC. You can also reference any private synonym contained in another schema if you have been granted the necessary object privileges for the private synonym.

You can only reference another user's synonym using the object privileges that you have been granted. For example, if you have the SELECT privilege for the jward.emp_tab synonym, then you can query the jward.emp_tab synonym, but you cannot insert rows using the synonym for jward.emp_tab.

A synonym can be referenced in a DML statement the same way that the underlying object of the synonym can be referenced. For example, if a synonym named emp_tab refers to a table or view, then the following statement is valid:

```
INSERT INTO Emp_tab (Empno, Ename, Job)
    VALUES (Emp_sequence.NEXTVAL, 'SMITH', 'CLERK');
```

If the synonym named fire_emp refers to a standalone procedure or package procedure, then you could execute it with the command

```
EXECUTE Fire_emp(7344);
```

Dropping Synonyms

You can drop any private synonym in your own schema. To drop a private synonym in another user's schema, you must have the DROP ANY SYNONYM system privilege. To drop a public synonym, you must have the DROP PUBLIC SYNONYM system privilege.

Drop a synonym that is no longer required using DROP SYNONYM statement. To drop a private synonym, omit the PUBLIC keyword. To drop a public synonym, include the PUBLIC keyword.

For example, the following statement drops the private synonym named emp:

```
DROP SYNONYM emp;
```

The following statement drops the public synonym named `public_emp`:

```
DROP PUBLIC SYNONYM public_emp;
```

When you drop a synonym, its definition is removed from the data dictionary. All objects that reference a dropped synonym remain. However, they become invalid (not usable). For more information about how dropping synonyms can affect other schema objects, see "Managing Object Dependencies".

> **See Also:** *Oracle Database SQL Reference* for syntax and additional information about the `DROP SYNONYM` statement

Viewing Information About Views, Synonyms, and Sequences

The following views display information about views, synonyms, and sequences:

View	Description

`DBA_VIEWS`
`ALL_VIEWS`
`USER_VIEWS`
	`DBA` view describes all views in the database. `ALL` view is restricted to views accessible to the current user. `USER` view is restricted to views owned by the current user.

`DBA_SYNONYMS`
`ALL_SYNONYMS`
`USER_SYNONYMS`
	These views describe synonyms.

`DBA_SEQUENCES`
`ALL_SEQUENCES`
`USER_SEQUENCES`
	These views describe sequences.

`DBA_UPDATABLE_COLUMNS`
`ALL_UPDATABLE_COLUMNS`
`USER_UPDATABLE_COLUMNS`
	These views describe all columns in join views that are updatable.

> **See Also:** *Oracle Database Reference* for complete descriptions of these views

20

General Management of Schema Objects

This chapter describes schema object management issues that are common across multiple types of schema objects. The following topics are presented:

Creating Multiple Tables and Views in a Single Operation

You can create several tables and views and grant privileges in one operation using the `CREATE SCHEMA` statement. The `CREATE SCHEMA` statement is useful if you want to guarantee the creation of several tables, views, and grants in one operation. If an individual table, view or grant fails, the entire statement is rolled back. None of the objects are created, nor are the privileges granted.

Specifically, the `CREATE SCHEMA` statement can include *only* `CREATE TABLE`, `CREATE VIEW`, and `GRANT` statements. You must have the privileges necessary to issue the included statements. You are not actually creating a schema, that is done

when the user is created with a CREATE USER statement. Rather, you are populating the schema.

The following statement creates two tables and a view that joins data from the two tables:

```
CREATE SCHEMA AUTHORIZATION scott
    CREATE TABLE dept (
        deptno NUMBER(3,0) PRIMARY KEY,
        dname VARCHAR2(15),
        loc VARCHAR2(25))
    CREATE TABLE emp (
        empno NUMBER(5,0) PRIMARY KEY,
        ename VARCHAR2(15) NOT NULL,
        job VARCHAR2(10),
        mgr NUMBER(5,0),
        hiredate DATE DEFAULT (sysdate),
        sal NUMBER(7,2),
        comm NUMBER(7,2),
        deptno NUMBER(3,0) NOT NULL
        CONSTRAINT dept_fkey REFERENCES dept)
   CREATE VIEW sales_staff AS
        SELECT empno, ename, sal, comm
        FROM emp
        WHERE deptno = 30
        WITH CHECK OPTION CONSTRAINT sales_staff_cnst
        GRANT SELECT ON sales_staff TO human_resources;
```

The CREATE SCHEMA statement does not support Oracle Database extensions to the ANSI CREATE TABLE and CREATE VIEW statements, including the STORAGE clause.

> **See Also:** *Oracle Database SQL Reference* for syntax and other information about the CREATE SCHEMA statement

Analyzing Tables, Indexes, and Clusters

You analyze a schema object (table, index, or cluster) to:

- Collect and manage statistics for it
- Verify the validity of its storage format
- Identify migrated and chained rows of a table or cluster

Note: Do not use the COMPUTE and ESTIMATE clauses of ANALYZE to collect optimizer statistics. These clauses are supported for backward compatibility. Instead, use the DBMS_STATS package, which lets you collect statistics in parallel, collect global statistics for partitioned objects, and fine tune your statistics collection in other ways. The cost-based optimizer, which depends upon statistics, will eventually use only statistics that have been collected by DBMS_STATS. See *PL/SQL Packages and Types Reference* for more information on the DBMS_STATS package.

You must use the ANALYZE statement (rather than DBMS_STATS) for statistics collection not related to the cost-based optimizer, such as:

- To use the VALIDATE or LIST CHAINED ROWS clauses
- To collect information on freelist blocks

The following topics are discussed in this section:

Using DBMS_STATS to Collect Table and Index Statistics

You can use the DBMS_STATS package or the ANALYZE statement to gather statistics about the physical storage characteristics of a table, index, or cluster. These statistics are stored in the data dictionary and can be used by the optimizer to choose the most efficient execution plan for SQL statements accessing analyzed objects.

Oracle recommends using the more versatile DBMS_STATS package for gathering optimizer statistics, but you must use the ANALYZE statement to collect statistics unrelated to the optimizer, such as empty blocks, average space, and so forth.

The DBMS_STATS package allows both the gathering of statistics, including utilizing parallel execution, and the external manipulation of statistics. Statistics can be stored in tables outside of the data dictionary, where they can be manipulated without affecting the optimizer. Statistics can be copied between databases or backup copies can be made.

The following DBMS_STATS procedures enable the gathering of optimizer statistics:

- GATHER_INDEX_STATS
- GATHER_TABLE_STATS
- GATHER_SCHEMA_STATS
- GATHER_DATABASE_STATS

See Also:

- *Oracle Database Performance Tuning Guide* for information about using DBMS_STATS to gather statistics for the optimizer
- *PL/SQL Packages and Types Reference* for a description of the DBMS_STATS package

Validating Tables, Indexes, Clusters, and Materialized Views

To verify the integrity of the structure of a table, index, cluster, or materialized view, use the ANALYZE statement with the VALIDATE STRUCTURE option. If the structure is valid, no error is returned. However, if the structure is corrupt, you receive an error message.

For example, in rare cases such as hardware or other system failures, an index can become corrupted and not perform correctly. When validating the index, you can confirm that every entry in the index points to the correct row of the associated table. If the index is corrupt, you can drop and re-create it.

If a table, index, or cluster is corrupt, you should drop it and re-create it. If a materialized view is corrupt, perform a complete refresh and ensure that you have remedied the problem. If the problem is not corrected, drop and re-create the materialized view.

The following statement analyzes the emp table:

```
ANALYZE TABLE emp VALIDATE STRUCTURE;
```

You can validate an object and all related objects (for example, indexes) by including the CASCADE option. The following statement validates the emp table and all associated indexes:

```
ANALYZE TABLE emp VALIDATE STRUCTURE CASCADE;
```

You can specify that you want to perform structure validation online while DML is occurring against the object being validated. There can be a slight performance impact when validating with ongoing DML affecting the object, but this is offset by

the flexibility of being able to perform ANALYZE online. The following statement validates the emp table and all associated indexes online:

```
ANALYZE TABLE emp VALIDATE STRUCTURE CASCADE ONLINE;
```

Listing Chained Rows of Tables and Clusters

You can look at the chained and migrated rows of a table or cluster using the ANALYZE statement with the LIST CHAINED ROWS clause. The results of this statement are stored in a specified table created explicitly to accept the information returned by the LIST CHAINED ROWS clause. These results are useful in determining whether you have enough room for updates to rows. For example, this information can show whether PCTFREE is set appropriately for the table or cluster.

Creating a CHAINED_ROWS Table

To create the table to accept data returned by an ANALYZE ... LIST CHAINED ROWS statement, execute the UTLCHAIN.SQL or UTLCHN1.SQL script. These scripts are provided by the database. They create a table named CHAINED_ROWS in the schema of the user submitting the script.

> **Note:** Your choice of script to execute for creating the CHAINED_ROWS table is dependent upon the compatibility level of your database and the type of table you are analyzing. See the *Oracle Database SQL Reference* for more information.

After a CHAINED_ROWS table is created, you specify it in the INTO clause of the ANALYZE statement. For example, the following statement inserts rows containing information about the chained rows in the emp_dept cluster into the CHAINED_ROWS table:

```
ANALYZE CLUSTER emp_dept LIST CHAINED ROWS INTO CHAINED_ROWS;
```

> **See Also:** *Oracle Database Reference* for a description of the CHAINED_ROWS table

Eliminating Migrated or Chained Rows in a Table

You can use the information in the CHAINED_ROWS table to reduce or eliminate migrated and chained rows in an existing table. Use the following procedure.

1. Use the ANALYZE statement to collect information about migrated and chained rows.

```
ANALYZE TABLE order_hist LIST CHAINED ROWS;
```

2. Query the output table:

```
SELECT *
FROM CHAINED_ROWS
WHERE TABLE_NAME = 'ORDER_HIST';

OWNER_NAME  TABLE_NAME  CLUST... HEAD_ROWID          TIMESTAMP
----------  ----------  -----... ------------------  ---------
SCOTT       ORDER_HIST       ... AAAAluAAHAAAAA1AAA  04-MAR-96
SCOTT       ORDER_HIST       ... AAAAluAAHAAAAA1AAB  04-MAR-96
SCOTT       ORDER_HIST       ... AAAAluAAHAAAAA1AAC  04-MAR-96
```

   The output lists all rows that are either migrated or chained.

3. If the output table shows that you have many migrated or chained rows, then you can eliminate migrated rows by continuing through the following steps:

4. Create an intermediate table with the same columns as the existing table to hold the migrated and chained rows:

```
CREATE TABLE int_order_hist
   AS SELECT *
      FROM order_hist
      WHERE ROWID IN
         (SELECT HEAD_ROWID
            FROM CHAINED_ROWS
            WHERE TABLE_NAME = 'ORDER_HIST');
```

5. Delete the migrated and chained rows from the existing table:

```
DELETE FROM order_hist
   WHERE ROWID IN
      (SELECT HEAD_ROWID
         FROM CHAINED_ROWS
         WHERE TABLE_NAME = 'ORDER_HIST');
```

6. Insert the rows of the intermediate table into the existing table:

```
INSERT INTO order_hist
   SELECT *
   FROM int_order_hist;
```

7. Drop the intermediate table:

```
DROP TABLE int_order_history;
```

8. Delete the information collected in step 1 from the output table:

```
DELETE FROM CHAINED_ROWS
   WHERE TABLE_NAME = 'ORDER_HIST';
```

9. Use the ANALYZE statement again, and query the output table.

Any rows that appear in the output table are chained. You can eliminate chained rows only by increasing your data block size. It might not be possible to avoid chaining in all situations. Chaining is often unavoidable with tables that have a LONG column or long CHAR or VARCHAR2 columns.

Truncating Tables and Clusters

You can delete all rows of a table or all rows in a group of clustered tables so that the table (or cluster) still exists, but is completely empty. For example, consider a table that contains monthly data, and at the end of each month, you need to empty it (delete all rows) after archiving its data.

To delete all rows from a table, you have the following options:

- Use the DELETE statement.
- Use the DROP and CREATE statements.
- Use the TRUNCATE statement.

These options are discussed in the following sections

Using DELETE

You can delete the rows of a table using the DELETE statement. For example, the following statement deletes all rows from the emp table:

```
DELETE FROM emp;
```

If there are many rows present in a table or cluster when using the DELETE statement, significant system resources are consumed as the rows are deleted. For example, CPU time, redo log space, and undo segment space from the table and any associated indexes require resources. Also, as each row is deleted, triggers can be fired. The space previously allocated to the resulting empty table or cluster remains associated with that object. With DELETE you can choose which rows to delete, whereas TRUNCATE and DROP affect the entire object.

See Also: *Oracle Database SQL Reference* for syntax and other information about the DELETE statement

Using DROP and CREATE

You can drop a table and then re-create the table. For example, the following statements drop and then re-create the emp table:

```
DROP TABLE emp;
CREATE TABLE emp ( ... );
```

When dropping and re-creating a table or cluster, all associated indexes, integrity constraints, and triggers are also dropped, and all objects that depend on the dropped table or clustered table are invalidated. Also, all grants for the dropped table or clustered table are dropped.

Using TRUNCATE

You can delete all rows of the table using the TRUNCATE statement. For example, the following statement truncates the emp table:

```
TRUNCATE TABLE emp;
```

Using the TRUNCATE statement provides a fast, efficient method for deleting all rows from a table or cluster. A TRUNCATE statement does not generate any undo information and it commits immediately. It is a DDL statement and cannot be rolled back. A TRUNCATE statement does not affect any structures associated with the table being truncated (constraints and triggers) or authorizations. A TRUNCATE statement also specifies whether space currently allocated for the table is returned to the containing tablespace after truncation.

You can truncate any table or cluster in your own schema. Any user who has the DROP ANY TABLE system privilege can truncate a table or cluster in any schema.

Before truncating a table or clustered table containing a parent key, all referencing foreign keys in different tables must be disabled. A self-referential constraint does not have to be disabled.

As a TRUNCATE statement deletes rows from a table, triggers associated with the table are not fired. Also, a TRUNCATE statement does not generate any audit information corresponding to DELETE statements if auditing is enabled. Instead, a single audit record is generated for the TRUNCATE statement being issued. See the *Oracle Database Security Guide* for information about auditing.

A hash cluster cannot be truncated, nor can tables within a hash or index cluster be individually truncated. Truncation of an index cluster deletes all rows from all tables in the cluster. If all the rows must be deleted from an individual clustered table, use the DELETE statement or drop and re-create the table.

The REUSE STORAGE or DROP STORAGE options of the TRUNCATE statement control whether space currently allocated for a table or cluster is returned to the containing tablespace after truncation. The default option, DROP STORAGE, reduces the number of extents allocated to the resulting table to the original setting for MINEXTENTS. Freed extents are then returned to the system and can be used by other objects.

Alternatively, the REUSE STORAGE option specifies that all space currently allocated for the table or cluster remains allocated to it. For example, the following statement truncates the emp_dept cluster, leaving all extents previously allocated for the cluster available for subsequent inserts and deletes:

```
TRUNCATE CLUSTER emp_dept REUSE STORAGE;
```

The REUSE or DROP STORAGE option also applies to any associated indexes. When a table or cluster is truncated, all associated indexes are also truncated. The storage parameters for a truncated table, cluster, or associated indexes are not changed as a result of the truncation.

> **See Also:** *Oracle Database SQL Reference* for syntax and other information about the TRUNCATE TABLE and TRUNCATE CLUSTER statements

Enabling and Disabling Triggers

Database triggers are procedures that are stored in the database and activated ("fired") when specific conditions occur, such as adding a row to a table. You can use triggers to supplement the standard capabilities of the database to provide a highly customized database management system. For example, you can create a trigger to restrict DML operations against a table, allowing only statements issued during regular business hours.

Database triggers can be associated with a table, schema, or database. They are implicitly fired when:

- DML statements are executed (INSERT, UPDATE, DELETE) against an associated table

- Certain DDL statements are executed (for example: ALTER, CREATE, DROP) on objects within a database or schema
- A specified database event occurs (for example: STARTUP, SHUTDOWN, SERVERERROR)

This is not a complete list. See the *Oracle Database SQL Reference* for a full list of statements and database events that cause triggers to fire

Create triggers with the CREATE TRIGGER statement. They can be defined as firing BEFORE or AFTER the triggering event, or INSTEAD OF it. The following statement creates a trigger scott.emp_permit_changes on table scott.emp. The trigger fires before any of the specified statements are executed.

```
CREATE TRIGGER scott.emp_permit_changes
     BEFORE
     DELETE OR INSERT OR UPDATE
     ON scott.emp
     .
     .
     .
pl/sql block
     .
     .
     .
```

You can later remove a trigger from the database by issuing the DROP TRIGGER statement.

A trigger can be in either of two distinct modes:

- Enabled

  An enabled trigger executes its trigger body if a triggering statement is issued and the trigger restriction, if any, evaluates to true. By default, triggers are enabled when first created.

- Disabled

  A disabled trigger does not execute its trigger body, even if a triggering statement is issued and the trigger restriction (if any) evaluates to true.

To enable or disable triggers using the ALTER TABLE statement, you must own the table, have the ALTER object privilege for the table, or have the ALTER ANY TABLE system privilege. To enable or disable an individual trigger using the ALTER TRIGGER statement, you must own the trigger or have the ALTER ANY TRIGGER system privilege.

See Also:

- *Oracle Database Concepts* for a more detailed description of triggers
- *Oracle Database SQL Reference* for syntax of the CREATE TRIGGER statement
- *Oracle Database Application Developer's Guide - Fundamentals* for information about creating and using triggers

Enabling Triggers

You enable a disabled trigger using the ALTER TRIGGER statement with the ENABLE option. To enable the disabled trigger named reorder on the inventory table, enter the following statement:

```
ALTER TRIGGER reorder ENABLE;
```

To enable all triggers defined for a specific table, use the ALTER TABLE statement with the ENABLE ALL TRIGGERS option. To enable all triggers defined for the INVENTORY table, enter the following statement:

```
ALTER TABLE inventory
    ENABLE ALL TRIGGERS;
```

See Also: *Oracle Database SQL Reference* for syntax and other information about the ALTER TRIGGER statement

Disabling Triggers

Consider temporarily disabling a trigger if one of the following conditions is true:

- An object that the trigger references is not available.
- You must perform a large data load and want it to proceed quickly without firing triggers.
- You are loading data into the table to which the trigger applies.

You disable a trigger using the ALTER TRIGGER statement with the DISABLE option. To disable the trigger reorder on the inventory table, enter the following statement:

```
ALTER TRIGGER reorder DISABLE;
```

You can disable all triggers associated with a table at the same time using the `ALTER TABLE` statement with the `DISABLE ALL TRIGGERS` option. For example, to disable all triggers defined for the `inventory` table, enter the following statement:

```
ALTER TABLE inventory
    DISABLE ALL TRIGGERS;
```

Managing Integrity Constraints

Integrity constraints are rules that restrict the values for one or more columns in a table. Constraint clauses can appear in either `CREATE TABLE` or `ALTER TABLE` statements, and identify the column or columns affected by the constraint and identify the conditions of the constraint.

This section discusses the concepts of constraints and identifies the SQL statements used to define and manage integrity constraints. The following topics are contained in this section:

- Integrity Constraint States
- Setting Integrity Constraints Upon Definition
- Modifying, Renaming, or Dropping Existing Integrity Constraints
- Deferring Constraint Checks
- Reporting Constraint Exceptions
- Viewing Constraint Information

See Also:

- *Oracle Database Concepts* for a more thorough discussion of integrity constraints
- *Oracle Database Application Developer's Guide - Fundamentals* for detailed information and examples of using integrity constraints in applications

Integrity Constraint States

You can specify that a constraint is enabled (`ENABLE`) or disabled (`DISABLE`). If a constraint is enabled, data is checked as it is entered or updated in the database, and data that does not conform to the constraint is prevented from being entered. If

a constraint is disabled, then data that does not conform can be allowed to enter the database.

Additionally, you can specify that existing data in the table must conform to the constraint (VALIDATE). Conversely, if you specify NOVALIDATE, you are not ensured that existing data conforms.

An integrity constraint defined on a table can be in one of the following states:

- ENABLE, VALIDATE
- ENABLE, NOVALIDATE
- DISABLE, VALIDATE
- DISABLE, NOVALIDATE

For details about the meaning of these states and an understanding of their consequences, see the *Oracle Database SQL Reference*. Some of these consequences are discussed here.

Disabling Constraints

To enforce the rules defined by integrity constraints, the constraints should always be enabled. However, consider temporarily disabling the integrity constraints of a table for the following performance reasons:

- When loading large amounts of data into a table
- When performing batch operations that make massive changes to a table (for example, changing every employee's number by adding 1000 to the existing number)
- When importing or exporting one table at a time

In all three cases, temporarily disabling integrity constraints can improve the performance of the operation, especially in data warehouse configurations.

It is possible to enter data that violates a constraint while that constraint is disabled. Thus, you should always enable the constraint after completing any of the operations listed in the preceding bullet list.

Enabling Constraints

While a constraint is enabled, no row violating the constraint can be inserted into the table. However, while the constraint is disabled such a row can be inserted. This row is known as an exception to the constraint. If the constraint is in the enable novalidated state, violations resulting from data entered while the constraint was

disabled remain. The rows that violate the constraint must be either updated or deleted in order for the constraint to be put in the validated state.

You can identify exceptions to a specific integrity constraint while attempting to enable the constraint. See "Reporting Constraint Exceptions" on page 20-19. All rows violating constraints are noted in an `EXCEPTIONS` table, which you can examine.

Enable Novalidate Constraint State

When a constraint is in the enable novalidate state, all subsequent statements are checked for conformity to the constraint. However, any existing data in the table is not checked. A table with enable novalidated constraints can contain invalid data, but it is not possible to add new invalid data to it. Enabling constraints in the novalidated state is most useful in data warehouse configurations that are uploading valid OLTP data.

Enabling a constraint does not require validation. Enabling a constraint novalidate is much faster than enabling and validating a constraint. Also, validating a constraint that is already enabled does not require any DML locks during validation (unlike validating a previously disabled constraint). Enforcement guarantees that no violations are introduced during the validation. Hence, enabling without validating enables you to reduce the downtime typically associated with enabling a constraint.

Efficient Use of Integrity Constraints: A Procedure

Using integrity constraint states in the following order can ensure the best benefits:

1. Disable state.
2. Perform the operation (load, export, import).
3. Enable novalidate state.
4. Enable state.

Some benefits of using constraints in this order are:

- No locks are held.
- All constraints can go to enable state concurrently.
- Constraint enabling is done in parallel.
- Concurrent activity on table is permitted.

Setting Integrity Constraints Upon Definition

When an integrity constraint is defined in a CREATE TABLE or ALTER TABLE statement, it can be enabled, disabled, or validated or not validated as determined by your specification of the ENABLE / DISABLE clause. If the ENABLE / DISABLE clause is not specified in a constraint definition, the database automatically enables and validates the constraint.

Disabling Constraints Upon Definition

The following CREATE TABLE and ALTER TABLE statements both define and disable integrity constraints:

```
CREATE TABLE emp (
    empno NUMBER(5) PRIMARY KEY DISABLE,   . . . ;

ALTER TABLE emp
   ADD PRIMARY KEY (empno) DISABLE;
```

An ALTER TABLE statement that defines and disables an integrity constraint never fails because of rows in the table that violate the integrity constraint. The definition of the constraint is allowed because its rule is not enforced.

Enabling Constraints Upon Definition

The following CREATE TABLE and ALTER TABLE statements both define and enable integrity constraints:

```
CREATE TABLE emp (
    empno NUMBER(5) CONSTRAINT emp.pk PRIMARY KEY,   . . . ;

ALTER TABLE emp
    ADD CONSTRAINT emp.pk PRIMARY KEY (empno);
```

An ALTER TABLE statement that defines and attempts to enable an integrity constraint can fail because rows of the table violate the integrity constraint. If this case, the statement is rolled back and the constraint definition is not stored and not enabled.

When you enable a UNIQUE or PRIMARY KEY constraint an associated index is created.

Note: An efficient procedure for enabling a constraint that can make use of parallelism is described in "Efficient Use of Integrity Constraints: A Procedure" on page 20-14.

See Also: "Creating an Index Associated with a Constraint" on page 15-11

Modifying, Renaming, or Dropping Existing Integrity Constraints

You can use the ALTER TABLE statement to enable, disable, modify, or drop a constraint. When the database is using a UNIQUE or PRIMARY KEY index to enforce a constraint, and constraints associated with that index are dropped or disabled, the index is dropped, unless you specify otherwise.

While enabled foreign keys reference a PRIMARY or UNIQUE key, you cannot disable or drop the PRIMARY or UNIQUE key constraint or the index.

Disabling Enabled Constraints

The following statements disable integrity constraints. The second statement specifies that the associated indexes are to be kept.

```
ALTER TABLE dept
    DISABLE CONSTRAINT dname_ukey;

ALTER TABLE dept
    DISABLE PRIMARY KEY KEEP INDEX,
    DISABLE UNIQUE (dname, loc) KEEP INDEX;
```

The following statements enable novalidate disabled integrity constraints:

```
ALTER TABLE dept
    ENABLE NOVALIDATE CONSTRAINT dname_ukey;

ALTER TABLE dept
    ENABLE NOVALIDATE PRIMARY KEY,
    ENABLE NOVALIDATE UNIQUE (dname, loc);
```

The following statements enable or validate disabled integrity constraints:

```
ALTER TABLE dept
    MODIFY CONSTRAINT dname_key VALIDATE;

ALTER TABLE dept
```

```
    MODIFY PRIMARY KEY ENABLE NOVALIDATE;
```

The following statements enable disabled integrity constraints:

```
ALTER TABLE dept
    ENABLE CONSTRAINT dname_ukey;

ALTER TABLE dept
    ENABLE PRIMARY KEY,
    ENABLE UNIQUE (dname, loc);
```

To disable or drop a UNIQUE key or PRIMARY KEY constraint and all dependent FOREIGN KEY constraints in a single step, use the CASCADE option of the DISABLE or DROP clauses. For example, the following statement disables a PRIMARY KEY constraint and any FOREIGN KEY constraints that depend on it:

```
ALTER TABLE dept
    DISABLE PRIMARY KEY CASCADE;
```

Renaming Constraints

The ALTER TABLE ... RENAME CONSTRAINT statement enables you to rename any currently existing constraint for a table. The new constraint name must not conflict with any existing constraint names for a user.

The following statement renames the dname_ukey constraint for table dept:

```
ALTER TABLE dept
    RENAME CONSTRAINT dname_ukey TO dname_unikey;
```

When you rename a constraint, all dependencies on the base table remain valid.

The RENAME CONSTRAINT clause provides a means of renaming system generated constraint names.

Dropping Constraints

You can drop an integrity constraint if the rule that it enforces is no longer true, or if the constraint is no longer needed. You can drop the constraint using the ALTER TABLE statement with one of the following clauses:

- DROP PRIMARY KEY
- DROP UNIQUE
- DROP CONSTRAINT

The following two statements drop integrity constraints. The second statement keeps the index associated with the PRIMARY KEY constraint:

```
ALTER TABLE dept
    DROP UNIQUE (dname, loc);

ALTER TABLE emp
    DROP PRIMARY KEY KEEP INDEX,
    DROP CONSTRAINT dept_fkey;
```

If FOREIGN KEYs reference a UNIQUE or PRIMARY KEY, you must include the CASCADE CONSTRAINTS clause in the DROP statement, or you cannot drop the constraint.

Deferring Constraint Checks

When the database checks a constraint, it signals an error if the constraint is not satisfied. You can defer checking the validity of constraints until the end of a transaction.

When you issue the SET CONSTRAINTS statement, the SET CONSTRAINTS mode lasts for the duration of the transaction, or until another SET CONSTRAINTS statement resets the mode.

Notes:

- You cannot issue a SET CONSTRAINT statement inside a trigger.
- Deferrable unique and primary keys must use nonunique indexes.

Set All Constraints Deferred

Within the application being used to manipulate the data, you must set all constraints deferred before you actually begin processing any data. Use the following DML statement to set all deferrable constraints deferred:

```
SET CONSTRAINTS ALL DEFERRED;
```

> **Note:** The SET CONSTRAINTS statement applies only to the current transaction. The defaults specified when you create a constraint remain as long as the constraint exists. The ALTER SESSION SET CONSTRAINTS statement applies for the current session only.

Check the Commit (Optional)

You can check for constraint violations before committing by issuing the SET CONSTRAINTS ALL IMMEDIATE statement just before issuing the COMMIT. If there are any problems with a constraint, this statement fails and the constraint causing the error is identified. If you commit while constraints are violated, the transaction is rolled back and you receive an error message.

Reporting Constraint Exceptions

If exceptions exist when a constraint is validated, an error is returned and the integrity constraint remains novalidated. When a statement is not successfully executed because integrity constraint exceptions exist, the statement is rolled back. If exceptions exist, you cannot validate the constraint until all exceptions to the constraint are either updated or deleted.

To determine which rows violate the integrity constraint, issue the ALTER TABLE statement with the EXCEPTIONS option in the ENABLE clause. The EXCEPTIONS option places the rowid, table owner, table name, and constraint name of all exception rows into a specified table.

You must create an appropriate exceptions report table to accept information from the EXCEPTIONS option of the ENABLE clause before enabling the constraint. You can create an exception table by executing the UTLEXCPT.SQL script or the UTLEXPT1.SQL script.

> **Note:** Your choice of script to execute for creating the EXCEPTIONS table is dependent upon the compatibility level of your database and the type of table you are analyzing. See the *Oracle Database SQL Reference* for more information.

Both of these scripts create a table named EXCEPTIONS. You can create additional exceptions tables with different names by modifying and resubmitting the script.

The following statement attempts to validate the PRIMARY KEY of the dept table, and if exceptions exist, information is inserted into a table named EXCEPTIONS:

```
ALTER TABLE dept ENABLE PRIMARY KEY EXCEPTIONS INTO EXCEPTIONS;
```

If duplicate primary key values exist in the dept table and the name of the PRIMARY KEY constraint on dept is sys_c00610, then the following query will display those exceptions:

```
SELECT * FROM EXCEPTIONS;
```

The following exceptions are shown:

```
fROWID               OWNER      TABLE_NAME      CONSTRAINT
------------------  ---------  --------------  -----------
AAAAZ9AABAAABvqAAB  SCOTT      DEPT            SYS_C00610
AAAAZ9AABAAABvqAAG  SCOTT      DEPT            SYS_C00610
```

A more informative query would be to join the rows in an exception report table and the master table to list the actual rows that violate a specific constraint, as shown in the following statement and results:

```
SELECT deptno, dname, loc FROM dept, EXCEPTIONS
    WHERE EXCEPTIONS.constraint = 'SYS_C00610'
    AND dept.rowid = EXCEPTIONS.row_id;

DEPTNO     DNAME             LOC
---------- --------------    -----------
10         ACCOUNTING        NEW YORK
10         RESEARCH          DALLAS
```

All rows that violate a constraint must be either updated or deleted from the table containing the constraint. When updating exceptions, you must change the value violating the constraint to a value consistent with the constraint or to a null. After the row in the master table is updated or deleted, the corresponding rows for the exception in the exception report table should be deleted to avoid confusion with later exception reports. The statements that update the master table and the exception report table should be in the same transaction to ensure transaction consistency.

To correct the exceptions in the previous examples, you might issue the following transaction:

```
UPDATE dept SET deptno = 20 WHERE dname = 'RESEARCH';
DELETE FROM EXCEPTIONS WHERE constraint = 'SYS_C00610';
COMMIT;
```

When managing exceptions, the goal is to eliminate all exceptions in your exception report table.

> **Note:** While you are correcting current exceptions for a table with the constraint disabled, it is possible for other users to issue statements creating new exceptions. You can avoid this by marking the constraint `ENABLE NOVALIDATE` before you start eliminating exceptions.

> **See Also:** *Oracle Database Reference* for a description of the `EXCEPTIONS` table

Viewing Constraint Information

Oracle Database provides the following views that enable you to see constraint definitions on tables and to identify columns that are specified in constraints:

View	Description

`DBA_CONSTRAINTS`
`ALL_CONSTRAINTS`
`USER_CONSTRAINTS`
	`DBA` view describes all constraint definitions in the database. `ALL` view describes constraint definitions accessible to current user. `USER` view describes constraint definitions owned by the current user.

`DBA_CONS_COLUMNS`
`ALL_CONS_COLUMNS`
`USER_CONS_COLUMNS`
	`DBA` view describes all columns in the database that are specified in constraints. `ALL` view describes only those columns accessible to current user that are specified in constraints. `USER` view describes only those columns owned by the current user that are specified in constraints.

> **See Also:** *Oracle Database Reference* contains descriptions of the columns in these views

Renaming Schema Objects

To rename an object, it must be in your schema. You can rename schema objects in either of the following ways:

- Drop and re-create the object
- Rename the object using the `RENAME` statement

If you drop and re-create an object, all privileges granted for that object are lost. Privileges must be regranted when the object is re-created.

Alternatively, a table, view, sequence, or a private synonym of a table, view, or sequence can be renamed using the RENAME statement. When using the RENAME statement, integrity constraints, indexes, and grants made for the object are carried forward for the new name. For example, the following statement renames the `sales_staff` view:

```
RENAME sales_staff TO dept_30;
```

> **Note:** You cannot use RENAME for a stored PL/SQL program unit, public synonym, index, or cluster. To rename such an object, you must drop and re-create it.

Before renaming a schema object, consider the following effects:

- All views and PL/SQL program units dependent on a renamed object become invalid, and must be recompiled before next use.
- All synonyms for a renamed object return an error when used.

> **See Also:** *Oracle Database SQL Reference* for syntax of the RENAME statement

Managing Object Dependencies

This section describes the various object dependencies, and contains the following topics:

- Manually Recompiling Views
- Manually Recompiling Procedures and Functions
- Manually Recompiling Packages

First, review Table 20–1, which shows how objects are affected by changes to other objects on which they depend.

Table 20–1 Operations that Affect Object Status

Operation	Resulting Status of Object	Resulting Status of Dependent Objects
CREATE [TABLE|SEQUENCE|SYNONYM]	VALID if there are no errors	No change[1]
ALTER TABLE (ADD, RENAME, MODIFY columns) RENAME [TABLE|SEQUENCE|SYNONYM|VIEW]	VALID if there no errors	INVALID
DROP [TABLE|SEQUENCE|SYNONYM|VIEW|PROCEDURE|FUNCTION|PACKAGE]	None. The object is dropped.	INVALID
CREATE [VIEW|PROCEDURE][2]	VALID if there are no errors; INVALID if there are syntax or authorization errors	No change[1]
CREATE OR REPLACE [VIEW|PROCEDURE][2]	VALID if there are no errors; INVALID if there are syntax or authorization errors	INVALID
REVOKE *object privilege*[3] ON *object* TO|FROM *user*	No change	All objects of user that depend on object are INVALID[3]
REVOKE *object privilege*[3] ON *object* TO|FROM PUBLIC	No change	All objects in the database that depend on object are INVALID[3]
REVOKE *system privilege*[4] TO|FROM *user*	No change	All objects of user are INVALID[4]
REVOKE *system privilege*[4] TO|FROM PUBLIC	No change	All objects in the database are INVALID[4]

Notes to Table 20–1:

1. Can cause dependent objects to be made INVALID, if object did not exist earlier.
2. Standalone procedures and functions, packages, and triggers.

3. Only DML object privileges, including SELECT, INSERT, UPDATE, DELETE, and EXECUTE; revalidation does not require recompiling.
4. Only DML system privileges, including SELECT, INSERT, UPDATE, DELETE ANY TABLE, and EXECUTE ANY PROCEDURE; revalidation does not require recompiling.

The database automatically recompiles an invalid view or PL/SQL program unit the next time it is used. In addition, a user can force the database to recompile a view or program unit using the appropriate SQL statement with the COMPILE clause. Forced compilations are most often used to test for errors when a dependent view or program unit is invalid, but is not currently being used. In these cases, automatic recompilation would not otherwise occur until the view or program unit was executed. To identify invalid dependent objects, query the views USER/ALL/DBA_OBJECTS.

Manually Recompiling Views

To recompile a view manually, you must have the ALTER ANY TABLE system privilege or the view must be contained in your schema. Use the ALTER VIEW statement with the COMPILE clause to recompile a view. The following statement recompiles the view emp_dept contained in your schema:

```
ALTER VIEW emp_dept COMPILE;
```

> **See Also:** *Oracle Database SQL Reference* for syntax and other information about the ALTER VIEW statement

Manually Recompiling Procedures and Functions

To recompile a standalone procedure manually, you must have the ALTER ANY PROCEDURE system privilege or the procedure must be contained in your schema. Use the ALTER PROCEDURE/FUNCTION statement with the COMPILE clause to recompile a standalone procedure or function. The following statement recompiles the stored procedure update_salary contained in your schema:

```
ALTER PROCEDURE update_salary COMPILE;
```

> **See Also:** *Oracle Database SQL Reference* for syntax and other information about the ALTER PROCEDURE statement

Manually Recompiling Packages

To recompile a package manually, you must have the ALTER ANY PROCEDURE system privilege or the package must be contained in your schema. Use the ALTER PACKAGE statement with the COMPILE clause to recompile either a package body or both a package specification and body. The following statement recompiles just the body of the package acct_mgmt:

```
ALTER PACKAGE acct_mgmt COMPILE BODY;
```

The next statement compiles both the body and specification of the package acct_mgmt:

```
ALTER PACKAGE acct_mgmt COMPILE PACKAGE;
```

> **See Also:** *Oracle Database SQL Reference* for syntax and other information about the ALTER PACKAGE statement

Managing Object Name Resolution

Object names referenced in SQL statements can consist of several pieces, separated by periods. The following describes how the database resolves an object name.

1. Oracle Database attempts to qualify the first piece of the name referenced in the SQL statement. For example, in scott.emp, scott is the first piece. If there is only one piece, the one piece is considered the first piece.
   a. In the current schema, the database searches for an object whose name matches the first piece of the object name. If it does not find such an object, it continues with step b.
   b. The database searches for a public synonym that matches the first piece of the name. If it does not find one, it continues with step c.
   c. The database searches for a schema whose name matches the first piece of the object name. If it finds one, it returns to step b, now using the second piece of the name as the object to find in the qualified schema. If the second piece does not correspond to an object in the previously qualified schema or there is not a second piece, the database returns an error.

   If no schema is found in step c, the object cannot be qualified and the database returns an error.
2. A schema object has been qualified. Any remaining pieces of the name must match a valid part of the found object. For example, if scott.emp.deptno is the name, scott is qualified as a schema, emp is qualified as a table, and

deptno must correspond to a column (because emp is a table). If emp is qualified as a package, deptno must correspond to a public constant, variable, procedure, or function of that package.

When global object names are used in a distributed database, either explicitly or indirectly within a synonym, the local database resolves the reference locally. For example, it resolves a synonym to global object name of a remote table. The partially resolved statement is shipped to the remote database, and the remote database completes the resolution of the object as described here.

Because of how the database resolves references, it is possible for an object to depend on the nonexistence of other objects. This situation occurs when the dependent object uses a reference that would be interpreted differently were another object present. For example, assume the following:

- At the current point in time, the company schema contains a table named emp.
- A PUBLIC synonym named emp is created for company.emp and the SELECT privilege for company.emp is granted to the PUBLIC role.
- The jward schema does not contain a table or private synonym named emp.
- The user jward creates a view in his schema with the following statement:

```
CREATE VIEW dept_salaries AS
     SELECT deptno, MIN(sal), AVG(sal), MAX(sal) FROM emp
     GROUP BY deptno
     ORDER BY deptno;
```

When jward creates the dept_salaries view, the reference to emp is resolved by first looking for jward.emp as a table, view, or private synonym, none of which is found, and then as a public synonym named emp, which is found. As a result, the database notes that jward.dept_salaries depends on the nonexistence of jward.emp and on the existence of public.emp.

Now assume that jward decides to create a new view named emp in his schema using the following statement:

```
CREATE VIEW emp AS
     SELECT empno, ename, mgr, deptno
     FROM company.emp;
```

Notice that jward.emp does not have the same structure as company.emp.

As it attempts to resolve references in object definitions, the database internally makes note of dependencies that the new dependent object has on "nonexistent" objects--schema objects that, if they existed, would change the interpretation of the

object's definition. Such dependencies must be noted in case a nonexistent object is later created. If a nonexistent object is created, all dependent objects must be invalidated so that dependent objects can be recompiled and verified and all dependent function-based indexes must be marked unusable.

Therefore, in the previous example, as `jward.emp` is created, `jward.dept_salaries` is invalidated because it depends on `jward.emp`. Then when `jward.dept_salaries` is used, the database attempts to recompile the view. As the database resolves the reference to `emp`, it finds `jward.emp` (`public.emp` is no longer the referenced object). Because `jward.emp` does not have a `sal` column, the database finds errors when replacing the view, leaving it invalid.

In summary, you must manage dependencies on nonexistent objects checked during object resolution in case the nonexistent object is later created.

> **See Also:** "Schema Objects and Database Links" on page 29-21 for information about name resolution in a distributed database

Switching to a Different Schema

The following statement sets the schema of the current session to the schema name specified in the statement.

```
ALTER SESSION SET CURRENT_SCHEMA = <schema name>
```

In subsequent SQL statements, Oracle Database uses this schema name as the schema qualifier when the qualifier is omitted. In addition, the database uses the temporary tablespace of the specified schema for sorts, joins, and storage of temporary database objects. The session retains its original privileges and does not acquire any extra privileges by the preceding `ALTER SESSION` statement.

For example:

```
CONNECT scott/tiger
ALTER SESSION SET CURRENT_SCHEMA = joe;
SELECT * FROM emp_tab;
```

Since `emp_tab` is not schema-qualified, the table name is resolved under schema `joe`. But if `scott` does not have select privilege on table `joe.emp_tab`, then `scott` cannot execute the `SELECT` statement.

Displaying Information About Schema Objects

Oracle Database provides a PL/SQL package that enables you to determine the DDL that created an object and data dictionary views that you can use to display information about schema objects. Packages and views that are unique to specific types of schema objects are described in the associated chapters. This section describes views and packages that are generic in nature and apply to multiple schema objects.

Using a PL/SQL Package to Display Information About Schema Objects

The Oracle-supplied PL/SQL package DBMS_METADATA.GET_DDL lets you obtain metadata (in the form of DDL used to create the object) about a schema object.

> **See Also:** *PL/SQL Packages and Types Reference* for a description of PL/SQL packages

Example: Using the DBMS_METADATA Package

The DBMS_METADATA package is a powerful tool for obtaining the complete definition of a schema object. It enables you to obtain all of the attributes of an object in one pass. The object is described as DDL that can be used to (re)create it.

In the following statements the GET_DDL function is used to fetch the DDL for all tables in the current schema, filtering out nested tables and overflow segments. The SET_TRANSFORM_PARAM (with the handle value equal to DBMS_METADATA.SESSION_TRANSFORM meaning "for the current session") is used to specify that storage clauses are not to be returned in the SQL DDL. Afterwards, the session-level transform parameters are reset to their defaults. Once set, transform parameter values remain in effect until specifically reset to their defaults.

```
EXECUTE DBMS_METADATA.SET_TRANSFORM_PARAM(
     DBMS_METADATA.SESSION_TRANSFORM,'STORAGE',false);
SELECT DBMS_METADATA.GET_DDL('TABLE',u.table_name)
     FROM USER_ALL_TABLES u
     WHERE u.nested='NO'
     AND (u.iot_type is null or u.iot_type='IOT');
EXECUTE DBMS_METADATA.SET_TRANSFORM_PARAM(
     DBMS_METADATA.SESSION_TRANSFORM,'DEFAULT');
```

The output from DBMS_METADATA.GET_DDL is a LONG datatype. When using SQL*Plus, your output may be truncated by default. Issue the following SQL*Plus command before issuing the DBMS_METADATA.GET_DDL statement to ensure that your output is not truncated:

```
SQL> SET LONG 9999
```

> **See Also:** *Oracle XML Developer's Kit Programmer's Guide* for detailed information and further examples relating to the use of the DBMS_METADATA package

Using Views to Display Information About Schema Objects

These views display general information about schema objects:

View	Description
DBA_OBJECTS ALL_OBJECTS USER_OBJECTS	DBA view describes all schema objects in the database. ALL view describes objects accessible to current user. USER view describes objects owned by the current user.
DBA_CATALOG ALL_CATALOG USER_CATALOG	List the name, type, and owner (USER view does not display owner) for all tables, views, synonyms, and sequences in the database.
DBA_DEPENDENCIES ALL_DEPENDENCIES USER_DEPENDENCIES	Describe all dependencies between procedures, packages, functions, package bodies, and triggers, including dependencies on views without any database links.

The following sections contain examples of using some of these views.

> **See Also:** *Oracle Database Reference* for a complete description of data dictionary views

Example 1: Displaying Schema Objects By Type

The following query lists all of the objects owned by the user issuing the query:

```
SELECT OBJECT_NAME, OBJECT_TYPE
    FROM USER_OBJECTS;
```

The following is the query output:

```
OBJECT_NAME                    OBJECT_TYPE
-----------------------------  -------------------
EMP_DEPT                       CLUSTER
EMP                            TABLE
DEPT                           TABLE
```

```
EMP_DEPT_INDEX              INDEX
PUBLIC_EMP                  SYNONYM
EMP_MGR                     VIEW
```

Example 2: Displaying Dependencies of Views and Synonyms

When you create a view or a synonym, the view or synonym is based on its underlying base object. The ALL_DEPENDENCIES, USER_DEPENDENCIES, and DBA_DEPENDENCIES data dictionary views can be used to reveal the dependencies for a view. The ALL_SYNONYMS, USER_SYNONYMS, and DBA_SYNONYMS data dictionary views can be used to list the base object of a synonym. For example, the following query lists the base objects for the synonyms created by user jward:

```
SELECT TABLE_OWNER, TABLE_NAME, SYNONYM_NAME
    FROM DBA_SYNONYMS
    WHERE OWNER = 'JWARD';
```

The following is the query output:

```
TABLE_OWNER             TABLE_NAME   SYNONYM_NAME
----------------------  -----------  ----------------
SCOTT                   DEPT         DEPT
SCOTT                   EMP          EMP
```

21

Detecting and Repairing Data Block Corruption

This chapter explains using the DBMS_REPAIR PL/SQL package to repair data block corruption in database schema objects. It contains the following topics:

- Options for Repairing Data Block Corruption
- About the DBMS_REPAIR Package
- Using the DBMS_REPAIR Package
- DBMS_REPAIR Examples

> **Note:** If you are not familiar with the DBMS_REPAIR package, it is recommended that you work with an Oracle Support Services analyst when performing any of the repair procedures included in this package.

Options for Repairing Data Block Corruption

Oracle Database provides different methods for detecting and correcting data block corruption. One method of correction is to drop and re-create an object after the corruption is detected. However, this is not always possible or desirable. If data block corruption is limited to a subset of rows, another option is to rebuild the table by selecting all data except for the corrupt rows.

Yet another way to manage data block corruption is to use the DBMS_REPAIR package. You can use DBMS_REPAIR to detect and repair corrupt blocks in tables and indexes. Using this approach, you can address corruptions where possible, and also continue to use objects while you attempt to rebuild or repair them.

Note: Any corruption that involves the loss of data requires analysis and understanding of how that data fits into the overall database system. DBMS_REPAIR is not a magic wand. You must still determine whether the repair approach provided by this package is the appropriate tool for each specific corruption problem. Depending on the nature of the repair, you might lose data and logical inconsistencies can be introduced. Thus, you must weigh the gains and losses associated with using DBMS_REPAIR.

About the DBMS_REPAIR Package

This section describes the procedures contained in the DBMS_REPAIR package and notes some limitations and restrictions on their use.

See Also: *PL/SQL Packages and Types Reference* for more information on the syntax, restrictions, and exceptions for the DBMS_REPAIR procedures

DBMS_REPAIR Procedures

The following table lists the procedures included in the DBMS_REPAIR package.

Procedure Name	Description
CHECK_OBJECT	Detects and reports corruptions in a table or index
FIX_CORRUPT_BLOCKS	Marks blocks (that were previously identified by the CHECK_OBJECT procedure) as software corrupt
DUMP_ORPHAN_KEYS	Reports index entries (into an orphan key table) that point to rows in corrupt data blocks
REBUILD_FREELISTS	Rebuilds the free lists of the object
SEGMENT_FIX_STATUS	Provides the capability to fix the corrupted state of a bitmap entry when segment space management is AUTO
SKIP_CORRUPT_BLOCKS	When used, ignores blocks marked corrupt during table and index scans. If not used, you get error ORA-1578 when encountering blocks marked corrupt.
ADMIN_TABLES	Provides administrative functions (create, drop, purge) for repair or orphan key tables. **Note:** These tables are always created in the SYS schema.

These procedures are further described, with examples of their use, in "DBMS_ REPAIR Examples" on page 21-8.

Limitations and Restrictions

DBMS_REPAIR procedures have the following limitations:

- Tables with LOBs, nested tables, and varrays are supported, but the out of line columns are ignored.
- Clusters are supported in the SKIP_CORRUPT_BLOCKS and REBUILD_ FREELISTS procedures, but not in the CHECK_OBJECT procedure.
- Index-organized tables and LOB indexes are not supported.
- The DUMP_ORPHAN_KEYS procedure does not operate on bitmap indexes or function-based indexes.
- The DUMP_ORPHAN_KEYS procedure processes keys that are, at most, 3,950 bytes long.

Using the DBMS_REPAIR Package

The following approach is recommended when considering DBMS_REPAIR for addressing data block corruption:

- Task 1: Detect and Report Corruptions
- Task 2: Evaluate the Costs and Benefits of Using DBMS_REPAIR
- Task 3: Make Objects Usable
- Task 4: Repair Corruptions and Rebuild Lost Data

These tasks are discussed in succeeding sections.

Task 1: Detect and Report Corruptions

Your first task, before using DBMS_REPAIR, should be the detection and reporting of corruptions. Reporting not only indicates what is wrong with a block, but also identifies the associated repair directive. You have several options, in addition to DBMS_REPAIR, for detecting corruptions. Table 21–1 describes the different detection methodologies.

Table 21–1 Comparison of Corruption Detection Methods

Detection Method	Description
DBMS_REPAIR	Performs block checking for a specified table, partition, or index. Populates a repair table with results.
DB_VERIFY	External command-line utility that performs block checking on an offline database.
ANALYZE	Used with the VALIDATE STRUCTURE option, verifies the integrity of the structure of an index, table, or cluster; checks or verifies that your tables and indexes are in sync.
DB_BLOCK_CHECKING	Performed when the initialization parameter DB_BLOCK_CHECKING=TRUE. Identifies corrupt blocks before they actually are marked corrupt. Checks are performed when changes are made to a block.

DBMS_REPAIR: Using the CHECK_OBJECT and ADMIN_TABLES Procedures

The CHECK_OBJECT procedure checks and reports block corruptions for a specified object. Similar to the ANALYZE ... VALIDATE STRUCTURE statement for indexes and tables, block checking is performed for index and data blocks.

Not only does CHECK_OBJECT report corruptions, but it also identifies any fixes that would occur if FIX_CORRUPT_BLOCKS is subsequently run on the object. This information is made available by populating a repair table, which must first be created by the ADMIN_TABLES procedure.

After you run the CHECK_OBJECT procedure, a simple query on the repair table shows the corruptions and repair directives for the object. With this information, you can assess how best to address the problems reported.

DB_VERIFY: Performing an Offline Database Check

Typically, you use DB_VERIFY as an offline diagnostic utility when you encounter data corruption problems.

> **See Also:** *Oracle Database Utilities* for more information about DB_VERIFY

ANALYZE: Corruption Reporting

The ANALYZE TABLE ... VALIDATE STRUCTURE statement validates the structure of the analyzed object. If the database successfully validates the structure, a message confirming its validation is returned to you. If the database encounters

corruption in the structure of the object, an error message is returned to you. In this case, drop and re-create the object.

> **See Also:** *Oracle Database SQL Reference* for more information about the ANALYZE statement

DB_BLOCK_CHECKING (Block Checking Initialization Parameter)

You can set block checking for instances using the DB_BLOCK_CHECKING initialization parameter (the default value is FALSE). This checks data and index blocks whenever they are modified. DB_BLOCK_CHECKING is a dynamic parameter, modifiable by the ALTER SYSTEM SET statement. Block checking is always enabled for the system tablespace.

> **See Also:** *Oracle Database Reference* for more information about the DB_BLOCK_CHECKING initialization parameter

Task 2: Evaluate the Costs and Benefits of Using DBMS_REPAIR

Before using DBMS_REPAIR you must weigh the benefits of its use in relation to the liabilities. You should also examine other options available for addressing corrupt objects. Begin by answering the following questions:

- What is the extent of the corruption?

  To determine if there are corruptions and repair actions, execute the CHECK_OBJECT procedure, and query the repair table.

- What other options are available for addressing block corruptions? Consider the following:
  - If the data is available from another source, then drop, re-create, and repopulate the object.
  - Issue the CREATE TABLE ... AS SELECT statement from the corrupt table to create a new one.
  - Ignore the corruption by excluding corrupt rows from select statements.
  - Perform media recovery.
- What logical corruptions or side effects are introduced when you use DBMS_REPAIR to make an object usable? Can these be addressed? What is the effort required to do so?

It is possible that you do not have access to rows in blocks marked corrupt. However, a block could be marked corrupt even though there are still rows that you can validly access.

It is also possible that referential integrity constraints are broken when blocks are marked corrupt. If this occurs, disable and reenable the constraint; any inconsistencies are reported. After fixing all problems, you should be able to successfully reenable the constraint.

Logical corruption can occur when there are triggers defined on the table. For example, if rows are reinserted, should insert triggers be fired or not? You can address these issues only if you understand triggers and their use in your installation.

Free list blocks can become inaccessible. If a corrupt block is at the head or tail of a free list, space management reinitializes the free list. There then can be blocks that should be on a free list, but are not. You can address this by running the `REBUILD_FREELISTS` procedure.

Indexes and tables are out of sync. You can address this by first executing the `DUMP_ORPHAN_KEYS` procedure (to obtain information from the keys that might be useful in rebuilding corrupted data). Then issue the `ALTER INDEX ... REBUILD ONLINE` statement to get the table and its indexes back in sync.

- If repair involves loss of data, can this data be retrieved?

  You can retrieve data from the index when a data block is marked corrupt. The `DUMP_ORPHAN_KEYS` procedure can help you retrieve this information. Of course, retrieving data in this manner depends on the amount of redundancy between the indexes and the table.

Task 3: Make Objects Usable

In this task `DBMS_REPAIR` makes the object usable by ignoring corruptions during table and index scans.

Corruption Repair: Using the FIX_CORRUPT_BLOCKS and SKIP_CORRUPT_BLOCKS Procedures

You make a corrupt object usable by establishing an environment that skips corruptions that remain outside the scope of `DBMS_REPAIR` capabilities.

If corruptions involve a loss of data, such as a bad row in a data block, all such blocks are marked corrupt by the `FIX_CORRUPT_BLOCKS` procedure. Then, you can run the `SKIP_CORRUPT_BLOCKS` procedure, which skips blocks marked corrupt

for the object. When skip is set, table and index scans skip all blocks marked corrupt. This applies to both media and software corrupt blocks.

Implications when Skipping Corrupt Blocks

If an index and table are out of sync, then a SET TRANSACTION READ ONLY transaction can be inconsistent in situations where one query probes only the index, and then a subsequent query probes both the index and the table. If the table block is marked corrupt, then the two queries return different results, thereby breaking the rules of a read-only transaction. One way to approach this is to not skip corruptions when in a SET TRANSACTION READ ONLY transaction.

A similar issue occurs when selecting rows that are chained. Essentially, a query of the same row may or may not access the corruption, thereby producing different results.

Task 4: Repair Corruptions and Rebuild Lost Data

After making an object usable, you can perform the following repair activities.

Recover Data Using the DUMP_ORPHAN_KEYS Procedures

The DUMP_ORPHAN_KEYS procedure reports on index entries that point to rows in corrupt data blocks. All such index entries are inserted into an orphan key table that stores the key and rowid of the corruption.

After the index entry information has been retrieved, you can rebuild the index using the ALTER INDEX ... REBUILD ONLINE statement.

Repair Free Lists Using the REBUILD_FREELISTS Procedure

Use this procedure if free space in segments is being managed using free lists (SEGMENT SPACE MANAGEMENT MANUAL).

When a block marked "corrupt" is found at the head or tail of a free list, the free list is reinitialized and an error is returned. Although this takes the offending block off the free list, it causes you to lose free list access to all blocks that followed the corrupt block.

You can use the REBUILD_FREELISTS procedure to reinitialize the free lists. The object is scanned, and if it is appropriate for a block to be on the free list, it is added to the master free list. Free list groups are handled by distributing blocks in an equitable fashion, one block at a time. Any blocks marked "corrupt" in the object are ignored during the rebuild.

Fix Segment Bitmaps Using the SEGMENT_FIX_STATUS Procedure

Use this procedure if free space in segments is being managed using bitmaps (SEGMENT SPACE MANAGEMENT AUTO).

This procedure either recalculates the state of a bitmap entry based on the current contents of the corresponding block, or you can specify that a bitmap entry be set to a specific value. Usually, the state is recalculated correctly and there is no need to force a setting.

DBMS_REPAIR Examples

In this section, examples are presented reflecting the use of the DBMS_REPAIR procedures.

- Using ADMIN_TABLES to Build a Repair Table or Orphan Key Table
- Using the CHECK_OBJECT Procedure to Detect Corruption
- Fixing Corrupt Blocks with the FIX_CORRUPT_BLOCKS Procedure
- Finding Index Entries Pointing into Corrupt Data Blocks: DUMP_ORPHAN_KEYS
- Rebuilding Free Lists Using the REBUILD_FREELISTS Procedure
- Enabling or Disabling the Skipping of Corrupt Blocks: SKIP_CORRUPT_BLOCKS

Using ADMIN_TABLES to Build a Repair Table or Orphan Key Table

A repair table provides information about what corruptions were found by the CHECK_OBJECT procedure and how these will be addressed if the FIX_CORRUPT_BLOCKS procedure is run. Further, it is used to drive the execution of the FIX_CORRUPT_BLOCKS procedure.

An orphan key table is used when the DUMP_ORPHAN_KEYS procedure is executed and it discovers index entries that point to corrupt rows. The DUMP_ORPHAN_KEYS procedure populates the orphan key table by logging its activity and providing the index information in a usable manner.

The ADMIN_TABLE procedure is used to create, purge, or drop a repair table or an orphan key table.

Creating a Repair Table

The following example creates a repair table for the users tablespace.

```
BEGIN
DBMS_REPAIR.ADMIN_TABLES (
     TABLE_NAME => 'REPAIR_TABLE',
     TABLE_TYPE => dbms_repair.repair_table,
     ACTION     => dbms_repair.create_action,
     TABLESPACE => 'USERS');
END;
/
```

For each repair or orphan key table, a view is also created that eliminates any rows that pertain to objects that no longer exist. The name of the view corresponds to the name of the repair or orphan key table, but is prefixed by DBA_ (for example DBA_REPAIR_TABLE or DBA_ORPHAN_KEY_TABLE).

The following query describes the repair table created in the previous example.

```
DESC REPAIR_TABLE

 Name                          Null?    Type
 ----------------------------- -------- --------------
 OBJECT_ID                     NOT NULL NUMBER
 TABLESPACE_ID                 NOT NULL NUMBER
 RELATIVE_FILE_ID              NOT NULL NUMBER
 BLOCK_ID                      NOT NULL NUMBER
 CORRUPT_TYPE                  NOT NULL NUMBER
 SCHEMA_NAME                   NOT NULL VARCHAR2(30)
 OBJECT_NAME                   NOT NULL VARCHAR2(30)
 BASEOBJECT_NAME                        VARCHAR2(30)
 PARTITION_NAME                         VARCHAR2(30)
 CORRUPT_DESCRIPTION                    VARCHAR2(2000)
 REPAIR_DESCRIPTION                     VARCHAR2(200)
 MARKED_CORRUPT                NOT NULL VARCHAR2(10)
 CHECK_TIMESTAMP               NOT NULL DATE
 FIX_TIMESTAMP                          DATE
 REFORMAT_TIMESTAMP                     DATE
```

Creating an Orphan Key Table

This example illustrates the creation of an orphan key table for the `users` tablespace.

```
BEGIN
DBMS_REPAIR.ADMIN_TABLES (
     TABLE_NAME => 'ORPHAN_KEY_TABLE',
     TABLE_TYPE => dbms_repair.orphan_table,
     ACTION     => dbms_repair.create_action,
```

```
     TABLESPACE => 'USERS');
END;
/
```

The orphan key table is described in the following query:

```
DESC ORPHAN_KEY_TABLE

 Name                          Null?    Type
 ----------------------------- -------- -----------------
 SCHEMA_NAME                   NOT NULL VARCHAR2(30)
 INDEX_NAME                    NOT NULL VARCHAR2(30)
 IPART_NAME                             VARCHAR2(30)
 INDEX_ID                      NOT NULL NUMBER
 TABLE_NAME                    NOT NULL VARCHAR2(30)
 PART_NAME                              VARCHAR2(30)
 TABLE_ID                      NOT NULL NUMBER
 KEYROWID                      NOT NULL ROWID
 KEY                           NOT NULL ROWID
 DUMP_TIMESTAMP                NOT NULL DATE
```

Using the CHECK_OBJECT Procedure to Detect Corruption

The CHECK_OBJECT procedure checks the specified objects, and populates the repair table with information about corruptions and repair directives. You can optionally specify a range, partition name, or subpartition name when you would like to check a portion of an object.

Validation consists of checking all blocks in the object that have not previously been marked corrupt. For each block, the transaction and data layer portions are checked for self consistency. During CHECK_OBJECT, if a block is encountered that has a corrupt buffer cache header, then that block is skipped.

Here is an example of executing the CHECK_OBJECT procedure for the `scott.dept` table.

```
SET SERVEROUTPUT ON
DECLARE num_corrupt INT;
BEGIN
num_corrupt := 0;
DBMS_REPAIR.CHECK_OBJECT (
     SCHEMA_NAME => 'SCOTT',
     OBJECT_NAME => 'DEPT',
     REPAIR_TABLE_NAME => 'REPAIR_TABLE',
     CORRUPT_COUNT =>  num_corrupt);
DBMS_OUTPUT.PUT_LINE('number corrupt: ' || TO_CHAR (num_corrupt));
```

```
END;
/
```

SQL*Plus outputs the following line, indicating one corruption:

```
number corrupt: 1
```

Querying the repair table produces information describing the corruption and suggesting a repair action.

```
SELECT OBJECT_NAME, BLOCK_ID, CORRUPT_TYPE, MARKED_CORRUPT,
       CORRUPT_DESCRIPTION, REPAIR_DESCRIPTION
     FROM REPAIR_TABLE;

OBJECT_NAME                     BLOCK_ID CORRUPT_TYPE MARKED_COR
------------------------------ ---------- ------------ ----------
CORRUPT_DESCRIPTION
--------------------------------------------------------------------------------
REPAIR_DESCRIPTION
--------------------------------------------------------------------------------
DEPT                                    3            1 FALSE
kdbchk: row locked by non-existent transaction
        table=0   slot=0
        lockid=32   ktbbhitc=1
mark block software corrupt
```

At this point, the corrupted block has not yet been marked corrupt, so this is the time to extract any meaningful data. After the block is marked corrupt, the entire block must be skipped.

Fixing Corrupt Blocks with the FIX_CORRUPT_BLOCKS Procedure

Use the FIX_CORRUPT_BLOCKS procedure to fix the corrupt blocks in specified objects based on information in the repair table that was previously generated by the CHECK_OBJECT procedure. Prior to effecting any change to a block, the block is checked to ensure the block is still corrupt. Corrupt blocks are repaired by marking the block software corrupt. When a repair is performed, the associated row in the repair table is updated with a fix timestamp.

This example fixes the corrupt block in table scott.dept that was reported by the CHECK_OBJECT procedure.

```
SET SERVEROUTPUT ON
DECLARE num_fix INT;
BEGIN
```

```
num_fix := 0;
DBMS_REPAIR.FIX_CORRUPT_BLOCKS (
     SCHEMA_NAME => 'SCOTT',
     OBJECT_NAME=> 'DEPT',
     OBJECT_TYPE => dbms_repair.table_object,
     REPAIR_TABLE_NAME => 'REPAIR_TABLE',
     FIX_COUNT=> num_fix);
DBMS_OUTPUT.PUT_LINE('num fix: ' || TO_CHAR(num_fix));
END;
/

SQL*Plus outputs the following line:

num fix: 1
```

To following query confirms that the repair was done.

```
SELECT OBJECT_NAME, BLOCK_ID, MARKED_CORRUPT
     FROM REPAIR_TABLE;

OBJECT_NAME                     BLOCK_ID MARKED_COR
------------------------------ ---------- ----------
DEPT                                    3 TRUE
```

Finding Index Entries Pointing into Corrupt Data Blocks: DUMP_ORPHAN_KEYS

The DUMP_ORPHAN_KEYS procedure reports on index entries that point to rows in corrupt data blocks. For each such index entry encountered, a row is inserted into the specified orphan key table. The orphan key table must have been previously created.

This information can be useful for rebuilding lost rows in the table and for diagnostic purposes.

> **Note:** This should be run for every index associated with a table identified in the repair table.

In this example, `pk_dept` is an index on the `scott.dept` table. It is scanned to determine if there are any index entries pointing to rows in the corrupt data block.

```
SET SERVEROUTPUT ON
DECLARE num_orphans INT;
BEGIN
num_orphans := 0;
```

```
DBMS_REPAIR.DUMP_ORPHAN_KEYS (
     SCHEMA_NAME => 'SCOTT',
     OBJECT_NAME => 'PK_DEPT',
     OBJECT_TYPE => dbms_repair.index_object,
     REPAIR_TABLE_NAME => 'REPAIR_TABLE',
     ORPHAN_TABLE_NAME=> 'ORPHAN_KEY_TABLE',
     KEY_COUNT => num_orphans);
DBMS_OUTPUT.PUT_LINE('orphan key count: ' || TO_CHAR(num_orphans));
END;
/
```

The following line is output, indicating there are three orphan keys:

```
orphan key count: 3
```

Index entries in the orphan key table implies that the index should be rebuilt. This guarantees that a table probe and an index probe return the same result set.

Rebuilding Free Lists Using the REBUILD_FREELISTS Procedure

The `REBUILD_FREELISTS` procedure rebuilds the free lists for the specified object. All free blocks are placed on the master free list. All other free lists are zeroed. If the object has multiple free list groups, then the free blocks are distributed among all free lists, allocating to the different groups in round-robin fashion.

This example rebuilds the free lists for the table `scott.dept`.

```
BEGIN
DBMS_REPAIR.REBUILD_FREELISTS (
     SCHEMA_NAME => 'SCOTT',
     OBJECT_NAME => 'DEPT',
     OBJECT_TYPE => dbms_repair.table_object);
END;
/
```

Enabling or Disabling the Skipping of Corrupt Blocks: SKIP_CORRUPT_BLOCKS

The `SKIP_CORRUPT_BLOCKS` procedure enables or disables the skipping of corrupt blocks during index and table scans of the specified object. When the object is a table, skip applies to the table and its indexes. When the object is a cluster, it applies to all of the tables in the cluster, and their respective indexes.

The following example enables the skipping of software corrupt blocks for the `scott.dept` table:

```
BEGIN
DBMS_REPAIR.SKIP_CORRUPT_BLOCKS (
     SCHEMA_NAME => 'SCOTT',
     OBJECT_NAME => 'DEPT',
     OBJECT_TYPE => dbms_repair.table_object,
     FLAGS => dbms_repair.skip_flag);
END;
/
```

Querying `scott`'s tables using the `DBA_TABLES` view shows that `SKIP_CORRUPT` is enabled for table `scott.dept`.

```
SELECT OWNER, TABLE_NAME, SKIP_CORRUPT FROM DBA_TABLES
    WHERE OWNER = 'SCOTT';

OWNER                          TABLE_NAME                     SKIP_COR
------------------------------ ------------------------------ --------
SCOTT                          ACCOUNT                        DISABLED
SCOTT                          BONUS                          DISABLED
SCOTT                          DEPT                           ENABLED
SCOTT                          DOCINDEX                       DISABLED
SCOTT                          EMP                            DISABLED
SCOTT                          RECEIPT                        DISABLED
SCOTT                          SALGRADE                       DISABLED
SCOTT                          SCOTT_EMP                      DISABLED
SCOTT                          SYS_IOT_OVER_12255             DISABLED
SCOTT                          WORK_AREA                      DISABLED

10 rows selected.
```

Part V

Database Security

Part V addresses issues of user and privilege management affecting the security of the database. It includes the following chapters:

- Chapter 22, "Managing Users and Securing the Database"

22

Managing Users and Securing the Database

This chapter briefly discusses the creation and management of database users, with special attention to the importance of establishing security policies to protect your database, and provides cross-references to the appropriate security documentation.

The following topics are contained in this chapter:

The Importance of Establishing a Security Policy for Your Database

It is important to develop a security policy for every database. The security policy establishes methods for protecting your database from accidental or malicious destruction of data or damage to the database infrastructure.

Each database can have an administrator, referred to as the security administrator, who is responsible for implementing and maintaining the database security policy If the database system is small, the database administrator can have the responsibilities of the security administrator. However, if the database system is large, a designated person or group of people may have sole responsibility as security administrator.

For information about establishing security policies for your database, see *Oracle Database Security Guide*.

Managing Users and Resources

To connect to the database, each user must specify a valid user name that has been previously defined to the database. An account must have been established for the user, with information about the user being stored in the data dictionary.

When you create a database user (account), you specify the following attributes of the user:

- User name
- Authentication method
- Default tablespace
- Temporary tablespace
- Other tablespaces and quotas
- User profile

To learn how to create and manage users, see *Oracle Database Security Guide*.

Managing User Privileges and Roles

Privileges and roles are used to control user access to data and the types of SQL statements that can be executed. The table that follows describes the three types of privileges and roles:

Type	Description
System privilege	A system-defined privilege usually granted only by administrators. These privileges allow users to perform specific database operations.
Object privilege	A system-defined privilege that controls access to a specific object.
Role	A collection of privileges and other roles. Some system-defined roles exist, but most are created by administrators. Roles group together privileges and other roles, which facilitates the granting of multiple privileges and roles to users.

Privileges and roles can be granted to other users by users who have been granted the privilege to do so. The granting of roles and privileges starts at the administrator level. At database creation, the administrative user SYS is created and granted all system privileges and predefined Oracle Database roles. User SYS can

then grant privileges and roles to other users, and also grant those users the right to grant specific privileges to others.

To learn how to administer privileges and roles for users, see *Oracle Database Security Guide*.

Auditing Database Use

You can monitor and record selected user database actions, including those done by administrators. There are several reasons why you might want to implement database auditing. These, and descriptions of the types of auditing available to you, are described in *Oracle Database Security Guide*.

The *Oracle Database Security Guide* includes directions for enabling and configuring database auditing.

Part VI

Database Resource Management and Task Scheduling

Part VI discusses database resource management. It contains information about using the Oracle Database Resource Manager. Other chapters discuss Scheduler, a product with functionality designed to simplify your management of tasks, but also to provide a rich set of scheduling functionality to suit many needs.

This part contains the following chapters:

23

Managing Automatic System Tasks Using the Maintenance Window

Oracle Database is preconfigured to perform some routine database maintenance tasks so that you can run them at times when the system load is expected to be light. You can specify for such a time period a resource plan that controls the resource consumption of those maintenance tasks. When the designated time period ends, the database can switch to a different resource plan that lowers the resource allocation for any remaining maintenance tasks.

This chapter consists of the following sections:

Maintenance Windows

The Scheduler is a collection of functions and procedures packages in the `DBMS_SCHEDULER` package. The fully described beginning in Chapter 26, "Overview of Scheduler Concepts".

Two Scheduler windows are predefined upon installation of Oracle Database:

- `WEEKNIGHT_WINDOW` starts at 10 p.m. and ends at 6 a.m. every Monday through Friday.
- `WEEKEND_WINDOW` covers whole days Saturday and Sunday.

Together these windows constitute the `MAINTENANCE_WINDOW_GROUP` in which all system maintenance tasks are scheduled. Oracle Database uses the maintenance windows for automatic statistics collection and for some other internal system

maintenance jobs. If you are using the Resource Manager, you can also assign resource plans to these windows.

You can adjust the predefined maintenance windows to a time suitable to your database environment using the DBMS_SCHEDULER.SET_ATTRIBUTE procedure. For example, the following script moves the WEEKNIGHT_WINDOW to midnight to 8 a.m. every weekday morning:

```
EXECUTE DBMS_SCHEDULER.SET_ATTRIBUTE(
   'WEEKNIGHT_WINDOW',
   'repeat_interval',
   'freq=daily;byday=MON, TUE, WED, THU, FRI;byhour=0;byminute=0;bysecond=0');
```

You can also use the SET_ATTRIBUTE procedure to adjust any other property of a window. For example, the following script sets resource plan DEFAULT_MAINTENANCE_PLAN for the WEEKNIGHT_WINDOW:

```
EXECUTE DBMS_SCHEDULER.SET_ATTRIBUTE (
   'WEEKNIGHT_WINDOW',
   'resource_0plan',
   'DEFAULT_MAINTENANCE_PLAN');
```

If you have already enabled a different resource plan, Oracle Database will make the DEFAULT_MAINTENANCE_PLAN when the WEEKNIGHT_WINDOW opens, and will reactivate the original resource plan when the WEEKNIGHT_WINDOW closes.

> **See Also:** Chapter 27, "Using the Scheduler" for more information on the DBMS_SCHEDULER package and the SET_ATTRIBUTE procedure and *PL/SQL Packages and Types Reference* for reference information on the DBMS_SCHEDULER package

Automatic Statistics Collection Job

A Scheduler program GATHER_STATS_PROG and Scheduler job GATHER_STATS_JOB are predefined on installation of Oracle Database. GATHER_STATS_PROG collects optimizer statistics for all objects in the database for which there are no statistics or only stale statistics. GATHER_STATS_JOB is defined on GATHER_STATS_PROG and is scheduled to run in the MAINTENANCE_WINDOW_GROUP.

If you prefer to manage statistics collection manually, you can disable the job as follows:

```
EXECUTE DBMS_SCHEDULER.DISABLE('GATHER_STATS_JOB');
```

See Also: *Oracle Database Performance Tuning Guide* for more information on automatic statistics collection

Resource Management

A Resource Manager consumer group, AUTO_TASK_CONSUMER_GROUP, is predefined on installation of Oracle Database, and a Scheduler job class AUTO_TASKS_JOB_CLASS is defined based on this consumer group. The GATHER_STATS_JOB is defined to run in the AUTO_TASKS_JOB_CLASS job class.

When a resource plan is activated in the system, GATHER_STATS_JOB and any internal system tasks conform to the resource consumption specified for AUTO_TASK_CONSUMER_GROUP in this resource plan.

See Also: Chapter 24, "Using the Database Resource Manager" for more information about creating and modifying resource plans.

24

Using the Database Resource Manager

Oracle Database provides database resource management capability through its Database Resource Manager. This chapter introduces you to its use and contains the following topics:

Note: This chapter discusses the use of the Oracle-supplied `DBMS_RESOURCE_MANAGER` and `DBMS_RESOURCE_MANAGER_PRIVS` packages to administer the Database Resource Manager. You can more easily administer the Database Resource Manager through the Oracle Enterprise Manager (EM). It provides an easy to use graphical interface for administering the Database Resource Manager. See the Oracle Enterprise Manager documentation set for more information.

What Is the Database Resource Manager?

The main goal of the Database Resource Manager is to give the Oracle Database server more control over resource management decisions, thus circumventing problems resulting from inefficient operating system management.

This section contains the following topics:

What Problems Does the Database Resource Manager Address?

When database resource allocation decisions are left to the operating system, you may encounter the following problems:

- Excessive overhead

  Excessive overhead results from operating system context switching between Oracle Database server processes when the number of server processes is high.

- Inefficient scheduling

  The operating system deschedules database servers while they hold latches, which is inefficient.

- Inappropriate allocation of resources

  The operating system distributes resources equally among all active processes and is unable to prioritize one task over another.

- Inability to manage database-specific resources, such as parallel execution servers and active sessions

How Does the Database Resource Manager Address These Problems?

The Oracle Database Resource Manager helps to overcome these problems by allowing the database more control over how machine resources are allocated.

Specifically, using the Database Resource Manager, you can:

- Guarantee certain users a minimum amount of processing resources regardless of the load on the system and the number of users

- Distribute available processing resources by allocating percentages of CPU time to different users and applications. In a data warehouse, a higher percentage may be given to ROLAP (relational on-line analytical processing) applications than to batch jobs.
- Limit the degree of parallelism of any operation performed by members of a group of users
- Create an **active session pool**. This pool consists of a specified maximum number of user sessions allowed to be concurrently active within a group of users. Additional sessions beyond the maximum are queued for execution, but you can specify a timeout period, after which queued jobs will terminate.
- Allow **automatic switching** of users from one group to another group based on administrator defined criteria. If a member of a particular group of users creates a session that executes for longer than a specified amount of time, that session can be automatically switched to another group of users with different resource requirements.
- Prevent the execution of operations that the optimizer estimates will run for a longer time than a specified limit
- Create an **undo pool**. This pool consists of the amount of undo space that can be consumed in by a group of users.
- Limit the amount of time that a session can be idle. This can be further defined to mean only sessions that are blocking other sessions.
- Configure an instance to use a particular method of allocating resources. You can dynamically change the method, for example, from a daytime setup to a nighttime setup, without having to shut down and restart the instance.
- Allow the cancellation of long-running SQL statements and the termination of long-running sessions.

What Are the Elements of the Database Resource Manager?

The elements of database resource management, which you define through the Database Resource Manager packages, are described below.

Element	Description
Resource consumer group	User sessions grouped together based on resource processing requirements.

Element	Description
Resource plan	Contains directives that specify how resources are allocated to resource consumer groups.
Resource allocation method	The method / policy used by the Database Resource Manager when allocating for a particular resource; used by resource consumer groups and resource plans. The database provides the resource allocation methods that are available, but you determine which method to use.
Resource plan directive	Used by administrators to associate resource consumer groups with particular plans and allocate resources among resource consumer groups.

You will learn how to create and use these elements in later sections of this chapter.

Understanding Resource Plans

This section briefly introduces the concept of resource plans. Included are some illustrations of simple resource plans. More complex plans are included in the examples presented later ("Putting It All Together: Database Resource Manager Examples" on page 24-32), after it has been explained how to build and maintain the elements of the Database Resource Manager.

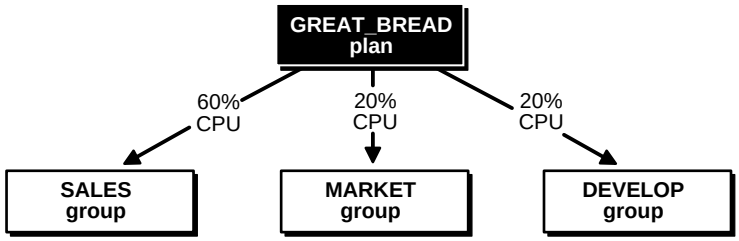
Resource plans specify the resource consumer groups belonging to the plan and contain directives for how resources are to be allocated among these groups. You use the DBMS_RESOURCE_MANAGER package to create and maintain these elements of the Database Resource Manager: resource plans, resource consumer groups, and resource plan directives. Plan information is stored in tables in the data dictionary. Several views are available for viewing plan data.

> **See Also:** "Viewing Database Resource Manager Information" on page 24-39

A Single-Level Resource Plan

The first illustration, shown in Figure 24–1, is of a single-level plan, where the plan allocates resources among resource consumer groups. The Great Bread Company has a plan called GREAT_BREAD that allocates CPU resources among three resource consumer groups. Specifically, SALES is allotted 60% of the CPU time, MARKET is allotted 20%, and DEVELOP receives the remaining 20%.

Figure 24–1 A Simple Resource Management Plan

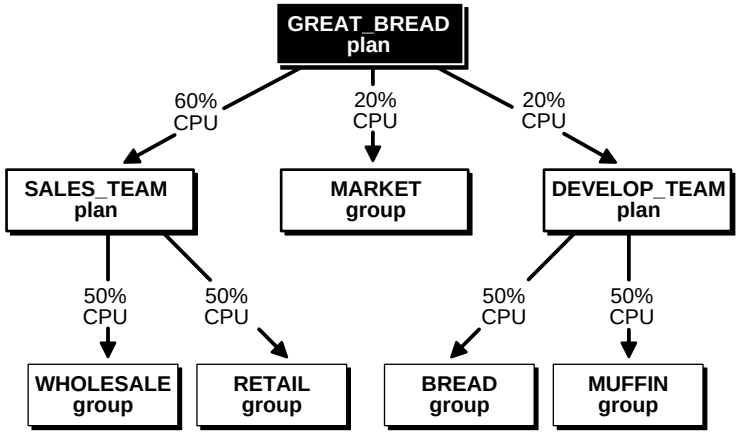


Oracle Database provides a procedure (CREATE_SIMPLE_PLAN) that enables you to quickly create a simple resource plan. This procedure is discussed in "Creating a Simple Resource Plan" on page 24-10.

A Multilevel Resource Plan

In addition to containing resource consumer groups, a plan can contain **subplans**. Perhaps the Great Bread Company chooses to divide their CPU resource as shown in Figure 24–2. The figure illustrates a **plan schema**, which contains a **top plan** (GREAT_BREAD) and all of its descendents.

Figure 24–2 A Multilevel Plan With Subplans



In this case, the GREAT_BREAD plan still allocates 20% of CPU resources to the consumer group MARKET. However, now it allocates CPU resources to subplans SALES_TEAM (60%), which in turn divides its share equally between the

WHOLESALE and RETAIL groups, and DEVELOP_TEAM (20%), which in turn divides its resources equally between the BREAD and MUFFIN groups.

It is possible for a subplan or consumer group to have more than one parent (owning plan), but there cannot be any loops in a plan schema. An example of a subplan having more that one parent would be if the Great Bread Company had a night plan and a day plan. Both the night plan and the day plan contain the sales subplan as a member, but perhaps with a different CPU resource allocation in each instance.

Note: As explained in subsequent sections, the plans described should also contain a plan directive for OTHER_GROUPS. To present a simplified view, however, this plan directive is not shown.

Resource Consumer Groups

Resource consumer groups are groups of users, or sessions, that are grouped together based on their processing needs. Resource plan directives, discussed next, specify how resources are allocated among consumer groups and subplans in a plan schema.

Resource Plan Directives

How resources are allocated to resource consumer groups is specified in resource allocation directives. The Database Resource Manager provides several means of allocating resources.

CPU Method This method enables you to specify how CPU resources are to be allocated among consumer groups or subplans. The multiple levels of CPU resource allocation (up to eight levels) provide a means of prioritizing CPU usage within a plan schema. Level 2 gets resources only after level 1 is unable to use all of its resources. Multiple levels not only provide a way of prioritizing, but they provide a way of explicitly specifying how all primary and leftover resources are to be used.

Active Session Pool with Queuing You can control the maximum number of concurrently active sessions allowed within a consumer group. This maximum designates the active session pool. When a session cannot be initiated because the pool is full, the session is placed into a queue. When an active session completes, the first session in the queue can then be scheduled for execution. You can also specify a timeout period after which a job in the execution queue (waiting for execution) will timeout, causing it to terminate with an error.

An entire parallel execution session is counted as one active session.

Degree of Parallelism Limit Specifying a parallel degree limit enables you to control the maximum degree of parallelism for any operation within a consumer group.

Automatic Consumer Group Switching This method enables you to control resources by specifying criteria that, if met, causes the automatic switching of sessions to another consumer group. The criteria used to determine switching are:

- Switch group: specifies the consumer group to which this session is switched if the other (following) criteria are met
- Switch time: specifies the length of time that a session can execute before it is switched to another consumer group
- Switch time in call: specifies the length of time that a session can execute before it is switched to another consumer group. Once the top call finishes, the session is restored to its original consumer group.
- Use estimate: specifies whether the database is to use its own estimate of how long an operation will execute

The Database Resource Manager switches a running session to *switch group* if the session is active for more than *switch time* seconds. Active means that the session is running and consuming resources, not waiting idly for user input or waiting for CPU cycles. The session is allowed to continue running, even if the active session pool for the new group is full. Under these conditions a consumer group can have more sessions running than specified by its active session pool. Once the session finishes its operation and becomes idle, it is switched back to its original group.

If *use estimate* is set to TRUE, the Database Resource Manager uses a predicted estimate of how long the operation will take to complete. If the database estimate is longer than the value specified as the switch time, then the database switches the session before execution starts. If this parameter is not set, the operation starts normally and only switches groups when other switch criteria are met.

Switch time in call is useful for three-tier applications where the middle tier server is using session pooling. At the end of every top call, a session is switched back to its original consumer group--that is, the group it would be in had it just logged in. A **top call** in PL/SQL is an entire PL/SQL block being treated as one call. A top call in SQL is an individual SQL statement issued separately by the client being treated as a one call.

You cannot specify both switch time in call and switch time.

Canceling SQL and Terminating Sessions You can also specify directives to cancel long-running SQL queries or to terminate long-running sessions. You specify this by setting `CANCEL_SQL` or `KILL_SESSION` as the switch group.

Execution Time Limit You can specify a maximum execution time allowed for an operation. If the database estimates that an operation will run longer than the specified maximum execution time, the operation is terminated with an error. This error can be trapped and the operation rescheduled.

Undo Pool You can specify an undo pool for each consumer group. An undo pool controls the amount of total undo that can be generated by a consumer group. When the total undo generated by a consumer group exceeds its undo limit, the current DML statement generating the redo is terminated. No other members of the consumer group can perform further data manipulation until undo space is freed from the pool.

Idle Time Limit You can specify an amount of time that a session can be idle, after which it will be terminated. You can further restrict such termination to only sessions that are blocking other sessions.

Administering the Database Resource Manager

You must have the system privilege `ADMINISTER_RESOURCE_MANAGER` to administer the Database Resource Manager. Typically, database administrators have this privilege with the `ADMIN` option as part of the DBA (or equivalent) role.

Being an administrator for the Database Resource Manager lets you execute all of the procedures in the `DBMS_RESOURCE_MANAGER` package. These are listed in the following table, and their use is explained in succeeding sections of this chapter.

Procedure	Description
`CREATE_SIMPLE_PLAN`	Creates a simple resource plan, containing up to eight consumer groups, in one step. This is the quickest way to get started when you use this package.
`CREATE_PLAN`	Creates a resource plan and specifies its allocation methods.
`UPDATE_PLAN`	Updates a resource plan.
`DELETE_PLAN`	Deletes a resource plan and its directives.
`DELETE_PLAN_CASCADE`	Deletes a resource plan and all of its descendents.
`CREATE_CONSUMER_GROUP`	Creates a resource consumer group.

Procedure	Description
UPDATE_CONSUMER_GROUP	Updates a consumer group.
DELETE_CONSUMER_GROUP	Deletes a consumer group.
CREATE_PLAN_DIRECTIVE	Specifies the resource plan directives that allocate resources to resource consumer groups within a plan or among subplans in a multilevel plan schema.
UPDATE_PLAN_DIRECTIVE	Updates plan directives
DELETE_PLAN_DIRECTIVE	Deletes plan directives
CREATE_PENDING_AREA	Creates a pending area (scratch area) within which changes can be made to a plan schema
VALIDATE_PENDING_AREA	Validates the pending changes to a plan schema
CLEAR_PENDING_AREA	Clears all pending changes from the pending area
SUBMIT_PENDING_AREA	Submits all changes for a plan schema
SET_INITIAL_CONSUMER_GROUP	Sets the initial consumer group for a user. This procedure has been deprecated. The database recommends that you use the SET_CONSUMER_GROUP_MAPPING procedure to specify the initial consumer group.
SWITCH_CONSUMER_GROUP_FOR_SESS	Switches the consumer group of a specific session
SWITCH_CONSUMER_GROUP_FOR_USER	Switches the consumer group of all sessions belonging to a specific user
SET_CONSUMER_GROUP_MAPPING	Maps sessions to consumer groups
SET_CONSUMER_MAPPING_PRI	Establishes session attribute mapping priorities

You may, as an administrator with the ADMIN option, choose to grant the administrative privilege to other users or roles. This is possible using the DBMS_RESOURCE_MANAGER_PRIVS package. This package contains the procedures listed in the table below.

Procedure	Description
GRANT_SYSTEM_PRIVILEGE	Grants ADMINISTER_RESOURCE_MANAGER system privilege to a user or role.
REVOKE_SYSTEM_PRIVILEGE	Revokes ADMINISTER_RESOURCE_MANAGER system privilege from a user or role.

Procedure	Description
GRANT_SWITCH_CONSUMER_GROUP	Grants permission to a user, role, or PUBLIC to switch to a specified resource consumer group.
REVOKE_SWITCH_CONSUMER_GROUP	Revokes permission for a user, role, or PUBLIC to switch to a specified resource consumer group.

The following example grants the administrative privilege to user scott, but does not grant scott the ADMIN option. Therefore, scott can execute all of the procedures in the DBMS_RESOURCE_MANAGER package, but scott cannot use the GRANT_SYSTEM_PRIVILEGE procedure to grant the administrative privilege to others.

```
EXEC DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SYSTEM_PRIVILEGE -
    (GRANTEE_NAME => 'scott', PRIVILEGE_NAME => 'ADMINISTER_RESOURCE_MANAGER', -
     ADMIN_OPTION => FALSE);
```

You can revoke this privilege using the REVOKE_SYSTEM_PRVILEGE procedure.

Note: The ADMINISTER_RESOURCE_MANAGER system privilege can only be granted or revoked by using the DBMS_RESOURCE_MANAGER_PRIVS package. It cannot be granted or revoked through the SQL GRANT or REVOKE statements.

The other procedures in the DBMS_RESOURCE_MANAGER_PRIVS package are discussed in "Managing the Switch Privilege" on page 24-26.

See Also: *PL/SQL Packages and Types Reference.* contains detailed information about the Database Resource Manager packages:

- DBMS_RESOURCE_MANAGER
- DBMS_RESOURCE_MANAGER_PRIVS

Creating a Simple Resource Plan

You can quickly create a simple resource plan that will be adequate for many situations using the CREATE_SIMPLE_PLAN procedure. This procedure enables you to create consumer groups and allocate resources to them by executing a single statement. Using this procedure, you are not required to invoke the procedures that

are described in succeeding sections for creating a pending area, creating each consumer group individually, and specifying resource plan directives.

You can specify the following parameters for the CREATE_SIMPLE_PLAN procedure:

Parameter	Description
SIMPLE_PLAN	Name of the plan
CONSUMER_GROUP1	Consumer group name for first group
GROUP1_CPU	CPU resource allocated to this group
CONSUMER_GROUP2	Consumer group name for second group
GROUP2_CPU	CPU resource allocated to this group
CONSUMER_GROUP3	Consumer group name for third group
GROUP3_CPU	CPU resource allocated to this group
CONSUMER_GROUP4	Consumer group name for fourth group
GROUP4_CPU	CPU resource allocated to this group
CONSUMER_GROUP5	Consumer group name for fifth group
GROUP5_CPU	CPU resource allocated to this group
CONSUMER_GROUP6	Consumer group name for sixth group
GROUP6_CPU	CPU resource allocated to this group
CONSUMER_GROUP7	Consumer group name for seventh group
GROUP7_CPU	CPU resource allocated to this group
CONSUMER_GROUP8	Consumer group name for eighth group
GROUP8_CPU	CPU resource allocated to this group

Up to eight consumer groups can be specified using this procedure and the only plan directive that can be specified is for CPU. The plan uses the EMPHASIS CPU allocation policy and each consumer group uses the ROUND_ROBIN scheduling policy.Each consumer group specified in the plan is allocated its CPU percentage at level 2. Also included in the plan are SYS_GROUP (a system-defined group that is the initial consumer group for the users SYS and SYSTEM) and OTHER_GROUPS.

Example: Using the CREATE_SIMPLE_PLAN Procedure

```
BEGIN
DBMS_RESOURCE_MANAGER.CREATE_SIMPLE_PLAN(SIMPLE_PLAN => 'simple_plan1',
   CONSUMER_GROUP1 => 'mygroup1', GROUP1_CPU => 80,
   CONSUMER_GROUP2 => 'mygroup2', GROUP2_CPU => 20);
END;
```

Executing the preceding statements creates the following plan:

Consumer Group	Level 1	Level 2	Level 3
SYS_GROUP	100%	-	-
mygroup1	-	80%	-
mygroup2	-	20%	-
OTHER_GROUPS	-	-	100%

Creating Complex Resource Plans

This section describes the actions and DBMS_RESOURCE_MANAGER procedures that you can use when your situation requires that you create more complex resource plans. It contains the following sections:

- Using the Pending Area for Creating Plan Schemas
- Creating Resource Plans
- Creating Resource Consumer Groups
- Specifying Resource Plan Directives

Using the Pending Area for Creating Plan Schemas

The first thing you must do to create or modify plan schemas is to create a **pending area**. This is a scratch area allowing you to stage your changes and to validate them before they are made active.

Creating a Pending Area

To create a pending area, you use the following statement:

```
EXEC DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA;
```

In effect, you are making the pending area active and "loading" all existing, or active, plan schemas into the pending area so that they can be updated or new plans added. Active plan schemas are those schemas already stored in the data dictionary for use by the Database Resource Manager. If you attempt to update a plan or add a new plan without first activating (creating) the pending area, you will receive an error message notifying you that the pending area is not active.

Views are available for inspecting all active resource plan schemas as well as the pending ones. These views are listed in Viewing Database Resource Manager Information on page 24-39.

Validating Changes

At any time when you are making changes in the pending area you can call the validate procedure as shown here.

```
EXEC DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA;
```

This procedure checks whether changes that have been made are valid. The following rules must be adhered to, and are checked by the validate procedure:

1. No plan schema can contain any loops.
2. All plans and resource consumer groups referred to by plan directives must exist.
3. All plans must have plan directives that point to either plans or resource consumer groups.
4. All percentages in any given level must not add up to greater than 100.
5. A plan that is currently being used as a top plan by an active instance cannot be deleted.
6. The following plan directive parameters can appear only in plan directives that refer to resource consumer groups (not other resource plans):
   - PARALLEL_DEGREE_LIMIT_P1
   - ACTIVE_SESS_POOL_P1
   - QUEUEING_P1
   - SWITCH_GROUP
   - SWITCH_TIME
   - SWITCH_ESTIMATE

- MAX_EST_EXEC_TIME
- UNDO_POOL
- MAX_IDLE_TIME
- MAX_IDLE_BLOCKER_TIME
- SWITCH_TIME_IN_CALL

7. There can be no more than 32 resource consumer groups in any active plan schema. Also, at most, a plan can have 32 children.
8. Plans and resource consumer groups cannot have the same name.
9. There must be a plan directive for OTHER_GROUPS somewhere in any active plan schema. This ensures that a session which is not part of any of the consumer groups included in the currently active plan is allocated resources (as specified by the OTHER_GROUPS directive).

You will receive an error message if any of the preceding rules are violated. You can then make changes to fix any problems and call the validate procedure again.

It is possible to create "orphan" consumer groups that have no plan directives referring to them. This allows the creation of consumer groups that will not currently be used, but may be part of some plan to be implemented in the future.

Submitting Changes

After you have validated your changes, call the submit procedure to make your changes active.

```
EXEC DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA;
```

The submit procedure also performs validation, so you do not necessarily need to make separate calls to the validate procedure. However, if you are making major changes to plan schemas, debugging problems is often easier if you incrementally validate your changes. No changes are submitted (made active) until validation is successful on all of the changes in the pending area.

The SUBMIT_PENDING_AREA procedure clears (deactivates) the pending area after successfully validating and committing the changes.

Note: A call to SUBMIT_PENDING_AREA may fail even if VALIDATE_PENDING_AREA succeeds. This can happen if, for example, a plan being deleted is loaded by an instance after a call to VALIDATE_PENDING_AREA, but before a call to SUBMIT_PENDING_AREA.

Clearing the Pending Area

There is also a procedure for clearing the pending area at any time. This statement causes all of your changes to be cleared from the pending area:

```
EXEC DBMS_RESOURCE_MANAGER.CLEAR_PENDING_AREA;
```

You must call the CREATE_PENDING_AREA procedure before you can again attempt to make changes.

Creating Resource Plans

When you create a resource plan, you can specify the parameters shown in the following table. The first parameter is required; the remainder are optional.

Parameter	Description
PLAN	Name of the plan.
COMMENT	Any descriptive comment.
CPU_MTH	Resource allocation method for specifying how much CPU each consumer group or subplan gets. EMPHASIS, the default method, is for multilevel plans that use percentages to specify how CPU is distributed among consumer groups. RATIO is for single-level plans that use ratios to specify how CPU is distributed.
ACTIVE_SESS_POOL_MTH	Active session pool resource allocation method. Limits the number of active sessions. All other sessions are inactive and wait in a queue to be activated. ACTIVE_SESS_POOL_ABSOLUTE is the default and only method available.
PARALLEL_DEGREE_LIMIT_MTH	Resource allocation method for specifying a limit on the degree of parallelism of any operation. PARALLEL_DEGREE_LIMIT_ABSOLUTE is the default and only method available.

Parameter	Description
QUEUEING_MTH	Queuing resource allocation method. Controls order in which queued inactive sessions will execute. FIFO_TIMEOUT is the default and only method available.

Oracle Database provides one resource plan, SYSTEM_PLAN, that contains a simple structure that may be adequate for some environments. It is illustrated later in "An Oracle-Supplied Plan" on page 24-35.

Creating a Plan

You create a plan using the CREATE_PLAN procedure. The following creates a plan called great_bread. You choose to use the default resource allocation methods.

```
EXEC DBMS_RESOURCE_MANAGER.CREATE_PLAN(PLAN => 'great_bread', -
    COMMENT => 'great plan');
```

Updating a Plan

Use the UPDATE_PLAN procedure to update plan information. If you do not specify the arguments for the UPDATE_PLAN procedure, they remain unchanged in the data dictionary. The following statement updates the COMMENT parameter.

```
EXEC DBMS_RESOURCE_MANAGER.UPDATE_PLAN(PLAN => 'great_bread', -
     NEW_COMMENT => 'great plan for great bread');
```

Deleting a Plan

The DELETE_PLAN procedure deletes the specified plan as well as all the plan directives associated with it. The following statement deletes the great_bread plan and its directives.

```
EXEC DBMS_RESOURCE_MANAGER.DELETE_PLAN(PLAN => 'great_bread');
```

The resource consumer groups themselves are not deleted, but they are no longer associated with the great_bread plan.

The DELETE_PLAN_CASCADE procedure deletes the specified plan as well as all its descendants (plan directives, subplans, resource consumer groups). If DELETE_PLAN_CASCADE encounters an error, it will roll back, leaving the plan schema unchanged.

Using the Ratio Policy

the RATIO policy is a single-level CPU allocation method. Instead of percentages, you specify numbers corresponding to the ratio of CPU you want to give to the consumer group. For example, given three consumer groups GOLD_CG, SILVER_CG, and BRONZE_CG, assume that we specify the following plan directives:

```
DBMS_RESOURCE_MANAGER.CREATE_PLAN
   (PLAN => 'service_level_plan',
    CPU_MTH -> 'RATIO',
    COMMENT => 'service level plan');
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE
   (PLAN => 'service_level_plan',
    GROUP_OR_SUBPLAN => 'GOLD_CG',
    COMMENT => 'Gold service level customers',
    CPU_P1 => 10);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE
   (PLAN => 'service_level_plan',
    GROUP_OR_SUBPLAN => 'SILVER_CG',
    COMMENT => 'Silver service level customers',
    CPU_P1 => 5);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE
   (PLAN => 'service_level_plan',
    GROUP_OR_SUBPLAN => 'BRONZE_CG',
    COMMENT => 'Bonze service level customers',
    CPU_P1 => 2);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE
    (PLAN => 'service_level_plan',
    GROUP_OR_SUBPLAN => 'OTHER_GROUPS',
    COMMENT => 'Lowest priority sessions',
    CPU_P1 => 1);
```

The ratio of CPU allocation would be 10:5:2:1 for the GOLD_CG, SILVER_CG, BRONZE_CG, and OTHER_GROUPS consumer groups, respectively.

If sessions exists only in the GOLD_CG and SILVER_CG consumer groups, then the ratio of CPU allocation would be 10:5 between the two groups.

Creating Resource Consumer Groups

When you create a resource consumer group, you can specify the following parameters:

Parameter	Description
CONSUMER_GROUP	Name of the consumer group.
COMMENT	Any comment.
CPU_MTH	The resource allocation method for distributing CPU among sessions in the consumer group. The default is ROUND_ROBIN, which uses a round-robin scheduler to ensure that sessions are fairly executed. RUN_TO_COMPLETION specifies that sessions with the largest active time are scheduled ahead of other sessions.

There are two special consumer groups that are always present in the data dictionary, and they cannot be modified or deleted. These are:

- DEFAULT_CONSUMER_GROUP

  This is the initial consumer group for all users / sessions that have not been explicitly assigned an initial consumer group. DEFAULT_CONSUMER_GROUP has switch privileges granted to PUBLIC; therefore, all users are automatically granted switch privilege for this consumer group (see "Managing the Switch Privilege" on page 24-26).

- OTHER_GROUPS

  This consumer group cannot be explicitly assigned to a user. OTHER_GROUPS must have a resource directive specified in the schema of any active plan. This group applies collectively to all sessions that belong to a consumer group that is not part of the currently active plan schema, including DEFAULT_CONSUMER_GROUP.

Additionally, two other groups, SYS_GROUP and LOW_GROUP, are provided as part of the Oracle-supplied SYSTEM_PLAN that is described in "An Oracle-Supplied Plan" on page 24-35.

Creating a Consumer Group

You create a consumer group using the CREATE_CONSUMER_GROUP procedure. The following creates a consumer group called sales. Remember, the pending area must be active to execute this statement successfully.

```
EXEC DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP (CONSUMER_GROUP => 'sales', -
    COMMENT => 'retail and wholesale sales');
```

Updating a Consumer Group

Use the UPDATE_CONSUMER_GROUP procedure to update consumer group information. If you do not specify the arguments for the UPDATE_CONSUMER_GROUP procedure, they remain unchanged in the data dictionary.

Deleting a Consumer Group

The DELETE_CONSUMER_GROUP procedure deletes the specified consumer group. Upon deletion of a consumer group, all users having the deleted group as their initial consumer group will have the DEFAULT_CONSUMER_GROUP set as their initial consumer group. All currently running sessions belonging to a deleted consumer group will be switched to DEFAULT_CONSUMER_GROUP.

Specifying Resource Plan Directives

Resource plan directives assign consumer groups to resource plans and provide the parameters for each resource allocation method. When you create a resource plan directive, you can specify the following parameters

Parameter	Description
PLAN	Name of the resource plan.
GROUP_OR_SUBPLAN	Name of the consumer group or subplan.
COMMENT	Any comment.
CPU_P1	For EMPHASIS, specifies the CPU percentage at the first level. For RATIO, specifies the weight of CPU usage. Default is NULL for all CPU parameters.
CPU_P2	For EMPHASIS, specifies CPU percentage at the second level. Not applicable for RATIO.
CPU_P3	For EMPHASIS, specifies CPU percentage at the third level. Not applicable for RATIO.
CPU_P4	For EMPHASIS, specifies CPU percentage at the fourth level. Not applicable for RATIO.
CPU_P5	For EMPHASIS, specifies CPU percentage at the fifth level. Not applicable for RATIO.
CPU_P6	For EMPHASIS, specifies CPU percentage at the sixth level. Not applicable for RATIO.

Parameter	Description
CPU_P7	For EMPHASIS, specifies CPU percentage at the seventh level. Not applicable for RATIO.
CPU_P8	For EMPHASIS, specifies CPU percentage at the eighth level. Not applicable for RATIO.
ACTIVE_SESS_POOL_P1	Specifies maximum number of concurrently active sessions for a consumer group. Default is UNLIMITED.
QUEUEING_P1	Specified time (in seconds) after which a job in an inactive session queue (waiting for execution) will time out. Default is UNLIMITED.
PARALLEL_DEGREE_LIMIT_P1	Specifies a limit on the degree of parallelism for any operation. Default is UNLIMITED.
SWITCH_GROUP	Specifies consumer group to which this session is switched if other switch criteria are met. If the group name is 'CANCEL_SQL', then the current call will be canceled when other switch criteria are met. If the group name is 'KILL_SESSION', then the session will be killed when other switch criteria are met. Default is NULL.
SWITCH_TIME	Specifies time (in seconds) that a session can execute before an action is taken. Default in UNLIMITED. You cannot specify both SWITCH_TIME and SWITCH_TIME_IN_CALL.
SWITCH_ESTIMATE	If TRUE, tells the database to use its execution time estimate to automatically switch the consumer group of an operation prior to beginning its execution. Default is FALSE.
MAX_EST_EXEC_TIME	Specifies the maximum execution time (in seconds) allowed for a session. If the optimizer estimates that an operation will take longer than MAX_EST_EXEC_TIME, the operation is not started and ORA-07455 is issued. If the optimizer does not provide an estimate, this directive has no effect. Default is UNLIMITED.
UNDO_POOL	Sets a maximum in kilobytes (K) on the total amount of undo generated by a consumer group. Default is UNLIMITED.
MAX_IDLE_TIME	Indicates the maximum session idle time. Default is NULL, which implies unlimited.

Parameter	Description
MAX_IDLE_BLOCKER_TIME	Indicates the maximum session idle time of a blocking session. Default is NULL, which implies unlimited.
SWITCH_TIME_IN_CALL	Specifies the time (in seconds) that a session can execute before an action is taken. At the end of the call, the consumer group of the session is restored to its original consumer group. Default is UNLIMITED. You cannot specify both SWITCH_TIME_IN_CALL and SWITCH_TIME.

Creating a Resource Plan Directive

You use the CREATE_PLAN_DIRECTIVE to create a resource plan directive. The following statement creates a resource plan directive for plan great_bread.

```
EXEC DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (PLAN => 'great_bread', -
     GROUP_OR_SUBPLAN => 'sales', COMMENT => 'sales group', -
     CPU_P1 => 60, PARALLEL_DEGREE_LIMIT_P1 => 4);
```

To complete the plan, similar to that shown in Figure 24–1, execute the following statements:

```
BEGIN
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (PLAN => 'great_bread',
     GROUP_OR_SUBPLAN => 'market', COMMENT => 'marketing group',
     CPU_P1 => 20);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (PLAN => 'great_bread',
     GROUP_OR_SUBPLAN => 'develop', COMMENT => 'development group',
     CPU_P1 => 20);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (PLAN => 'great_bread',
     GROUP_OR_SUBPLAN => 'OTHER_GROUPS', COMMENT => 'this one is required',
     CPU_P1 => 0, CPU_P2 => 100);
END;
```

In this plan, consumer group sales has a maximum degree of parallelism of 4 for any operation, while none of the other consumer groups are limited in their degree of parallelism. Also, whenever there are leftover level 1 CPU resources, they are allocated (100%) to OTHER_GROUPS.

Updating Resource Plan Directives

Use the UPDATE_PLAN_DIRECTIVE procedure to update plan directives. This example changes CPU allocation for resource consumer group develop.

```
EXEC DBMS_RESOURCE_MANAGER.UPDATE_PLAN_DIRECTIVE (PLAN => 'great_bread', -
     GROUP_OR_SUBPLAN => 'develop', NEW_CPU_P1 => 15);
```

If you do not specify the arguments for the UPDATE_PLAN_DIRECTIVE procedure, they remain unchanged in the data dictionary.

Deleting Resource Plan Directives

To delete a resource plan directive, use the DELETE_PLAN_DIRECTIVE procedure

How Resource Plan Directives Interact

If there are multiple resource plan directives that refer to the same consumer group, then the following rules apply for specific cases:

1. The parallel degree limit for the consumer group will be the *minimum* of all the incoming values.
2. The active session pool for the consumer group will be the *sum* of all the incoming values and the queue timeout will be the *minimum* of all incoming timeout values.
3. If there is more than one switch group and more than one switch time, the Database Resource Manager will choose the *most restrictive* of all incoming values. Specifically:
   - SWITCH_TIME = *min* (all incoming over_switch_time values)
   - SWITCH_ESTIMATE = TRUE overrides SWITCH_ESTIMATE = FALSE

   Notes:

   - If both plan directives specify the same switch time, but different switch groups, then the choice as to which group to switch to is statically but arbitrarily decided by the Database Resource Manager.
   - You cannot specify both SWITCH_TIME and SWITCH_TIME_IN_CALL plan directives for a single consumer group.
4. If a session is switched to another consumer group because it exceeds its switch time, that session will execute even if the active session pool for the new consumer group is full.

5. The maximum estimated execution time will be the most restrictive of all incoming values. Specifically:

   `MAX_EST_EXEC_TIME = min` (all incoming `MAX_EST_EXEC_TIME` values)

Managing Resource Consumer Groups

Before you enable the Database Resource Manager, you must assign resource consumer groups to users. This can be done manually, or you can provide mappings that enable the database to automatically assign user sessions to consumer groups depending upon session attributes.

In addition to providing procedures to create, update, or delete the elements used by the Database Resource Manager, the `DBMS_RESOURCE_MANAGER` package contains procedures that effect the assigning of resource consumer groups to users. It also provides procedures that allow you to temporarily switch a user session to another consumer group.

The `DBMS_RESOURCE_MANAGER_PRIVS` package, described earlier for granting the Database Resource Manager system privilege, can also be used to grant the switch privilege to another user, who can then alter their own consumer group.

Of the procedures discussed in this section, you use a pending area only for the `SET_CONSUMER_GROUP_MAPPING` and `SET_CONSUMER_MAPPING_PRI` procedures. These procedures are used for the automatic assigning of sessions to consumer groups.

This section contains the following topics:

- Assigning an Initial Resource Consumer Group
- Changing Resource Consumer Groups
- Managing the Switch Privilege
- Automatically Assigning Resource Consumer Groups to Sessions

Assigning an Initial Resource Consumer Group

The initial consumer group of a session is determined by mapping the attributes of the session to a consumer group. For more information on how to configure the mapping, see the section "Automatically Assigning Resource Consumer Groups to Sessions" on page 24-27.

Changing Resource Consumer Groups

There are two procedures, which are part of the DBMS_RESOURCE_MANAGER package, that allow administrators to change the resource consumer group of running sessions. Both of these procedures can also change the consumer group of any parallel execution server sessions associated with the coordinator session. The changes made by these procedures pertain to current sessions only; they are not persistent. They also do not change the initial consumer groups for users.

Instead of killing a session of a user who is using excessive CPU, an administrator can instead change that user's consumer group to one that is allowed less CPU. Or, this switching can be enforced automatically, using automatic consumer group switching resource plan directives.

Switching a Session

The SWITCH_CONSUMMER_GROUP_FOR_SESS causes the specified session to immediately be moved into the specified resource consumer group. In effect, this statement can raise or lower priority. The following statement changes the resource consumer group of a specific session to a new consumer group. The session identifier (SID) is 17, the session serial number (SERIAL#) is 12345, and the session is to be changed to the high_priority consumer group.

```
EXEC DBMS_RESOURCE_MANAGER.SWITCH_CONSUMER_GROUP_FOR_SESS ('17', '12345', -
   'high_priorty');
```

The SID, session serial number, and current resource consumer group for a session are viewable using the V$SESSION data dictionary view.

Switching Sessions for a User

The SWITCH_CONSUMER_GROUP_FOR_USER procedure changes the resource consumer group for all sessions with a given user name.

```
EXEC DBMS_RESOURCE_MANAGER.SWITCH_CONSUMER_GROUP_FOR_USER ('scott', -
    'low_group');
```

Using the DBMS_SESSION Package to Switch Consumer Group

If granted the switch privilege, users can switch their current consumer group using the SWITCH_CURRENT_CONSUMER_GROUP procedure in the DBMS_SESSION package.

This procedure enables users to switch to a consumer group for which they have the switch privilege. If the caller is another procedure, then this procedure enables

users to switch to a consumer group for which the owner of that procedure has switch privileges.

The parameters for this procedure are:

Parameter	Description
NEW_CONSUMER_GROUP	The consumer group to which the user is switching.
OLD_CONSUMER_GROUP	An output parameter. Stores the name of the consumer group from which the user switched. Can be used to switch back later.
INITIAL_GROUP_ON_ERROR	Controls behavior if a switching error occurs. If TRUE, in the event of an error, the user is switched to the initial consumer group. If FALSE, raise an error.

The following example illustrates switching to a new consumer group. By printing the value of the output parameter old_group, we illustrate how the old consumer group name has been saved.

```
SET serveroutput on
DECLARE
    old_group varchar2(30);
BEGIN
DBMS_SESSION.SWITCH_CURRENT_CONSUMER_GROUP('sales', old_group, FALSE);
DBMS_OUTPUT.PUT_LINE('OLD GROUP = ' || old_group);
END;
```

The following line is output:

```
OLD GROUP = DEFAULT_CONSUMER_GROUP
```

The DBMS_SESSION package can be used from within a PL/SQL application, thus allowing the application to change consumer groups, or effectively priority, dynamically.

Note: The Database Resource Manager also works in environments where a generic database user name is used to log on to an application. The DBMS_SESSION package can be called to switch the consumer group assignment of a session at session startup, or as particular modules are called.

See Also: *PL/SQL Packages and Types Reference* for additional examples and more information about the DBMS_SESSION package

Managing the Switch Privilege

Using the DBMS_RESOURCE_MANAGER_PRIVS package, you can grant or revoke the switch privilege to a user, role, or PUBLIC. The switch privilege gives users the privilege to switch their current resource consumer group to a specified resource consumer group. The package also enables you to revoke the switch privilege.

The actual switching is done by executing a procedure in the DBMS_SESSION package. A user who has been granted the switch privilege (or a procedure owned by that user) can use the SWITCH_CURRENT_CONSUMER_GROUP procedure to switch to another resource consumer group. The new group must be one to which the user has been specifically authorized to switch.

Granting the Switch Privilege

The following example grants the privilege to switch to a consumer group. User scott is granted the privilege to switch to consumer group bug_batch_group.

```
EXEC DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SWITCH_CONSUMER_GROUP ('scott', -
     'bug_batch_group', TRUE);
```

User scott is also granted permission to grant switch privileges for bug_batch_group to others.

If you grant a user permission to switch to a particular consumer group, then that user can switch their current consumer group to the new consumer group.

If you grant a role permission to switch to a particular resource consumer group, then any users who have been granted that role and have enabled that role can immediately switch their current consumer group to the new consumer group.

If you grant PUBLIC the permission to switch to a particular consumer group, then any user can switch to that group.

If the GRANT_OPTION argument is TRUE, then users granted switch privilege for the consumer group can also grant switch privileges for that consumer group to others.

Revoking Switch Privileges

The following example revokes user scott's privilege to switch to consumer group bug_batch_group.

```
EXEC DBMS_RESOURCE_MANAGER_PRIVS.REVOKE_SWITCH_CONSUMER_GROUP ('scott', -
     'bug_batch_group');
```

If you revoke a user's switch privileges to a particular consumer group, then any subsequent attempts by that user to switch to that consumer group will fail. If you revoke the initial consumer group from a user, then that user will automatically be part of the DEFAULT_CONSUMER_GROUP when logging in.

If you revoke from a role the switch privileges to a consumer group, then any users who only had switch privilege for the consumer group through that role will not be able to subsequently switch to that consumer group.

If you revoke switch privileges to a consumer group from PUBLIC, then any users other than those who are explicitly assigned switch privileges either directly or through PUBLIC, will not be able to subsequently switch to that consumer group.

Automatically Assigning Resource Consumer Groups to Sessions

You can configure the Database Resource Manager to automatically assign consumer groups to sessions by providing mappings between session attributes and consumer groups. Further, you can prioritize the mappings so as to indicate which mapping has precedence in case of conflicts.

There are two types of session attributes: login attributes and runtime attributes. The login attributes are meaningful only at session login time, when the Database Resource Manager determines the initial consumer group of the session. In contrast, a session that has already logged in can later be reassigned to another consumer group based on its run-time attributes.

You use the SET_CONSUMER_GROUP_MAPPING and SET_CONSUMER_MAPPING_PRI procedures to configure the automatic assigning of sessions to consumer groups. You must use a pending area for these procedures.

Creating Consumer Group Mappings

The SET_CONSUMER_GROUP_MAPPING procedure maps a session attribute to a consumer group. The parameters for this procedure are:

Parameter	Description
ATTRIBUTE	The login or runtime session attribute type
VALUE	The value of the attribute being mapped
CONSUMER_GROUP	The consumer group to be mapped to.

The attribute can be one of the following:

Attribute	Type	Description
ORACLE_USER	Login	The standard Oracle Database user name
SERVICE_NAME	Login	The service name used by the client to establish a login
CLIENT_OS_USER	Login	The operating system user name of the client that is logging in
CLIENT_PROGRAM	Login	The name of the client program used to log into the server
CLIENT_MACHINE	Login	The name of the machine from which the client is making the connection
MODULE_NAME	Runtime	The module name in the application currently executing as set by the DBMS_APPLICATION_INFO.SET_MODULE_NAME procedure, or the equivalent OCI attribute setting
MODULE_NAME_ACTION	Runtime	The current module and action being performed by the user as set by either of the following procedures, or their equivalent OCI attribute setting: ■ DBMS_APPLICATION_INFO.SET_MODULE_NAME ■ DBMS_APPLICATION_INFO.SET_ACTION The attribute is specified by the module name, followed by a period (.), followed by the action name.
SERVICE_MODULE	Runtime	A combination service and module names in this form: *service_name.module_name*
SERVICE_MODULE_ACTION	Runtime	A combination of service and module names and action name, in this form: *service_name.module_name.action_name*

Attribute	Type	Description
EXPLICIT	n/a	An explicit mapping refers to the consumer group explicitly requested by the client, for example by invoking the SWITCH_CURRENT_CONSUMER_GROUP, SWITCH_CONSUMER_GROUP_FOR_SESS, or SWITCH_CONSUMER_GROUP_FOR_USER procedure. See "Explicit Session Switching" on page 24-30

For each of the session attributes, you specify a mapping that consists of a set of pairs (attribute, consumer group) that determines how a session is matched to a consumer group. For example, the following statement causes user scott to map to the dev_group every time he logs in:

```
DBMS_RESOURCE_MANAGER.SET_CONSUMER_GROUP_MAPPING
     (DBMS_RESOURCE_MANAGER.ORACLE_USER, 'scott', 'dev group');
```

Creating Attribute Mapping Priorities

In order to decide between conflicting mappings, you can establish a priority ordering of the attributes from most important to least important. The priority of each attribute is set to a unique integer from 1 to 10 using the SET_CONSUMER_MAPPING_PRI procedure. For example, the following statement illustrates this setting of priorities:

```
DBMS_RESOURCE_MANAGER.SET_MAPPING_PRIORITY(
   EXPLICIT => 1, CLIENT_MACHINE => 2, MODULE_NAME => 3, ORACLE_USER => 4,
   SERVICE_NAME => 5, CLIENT_OS_USER => 6, CLIENT_PROGRAM => 7,
   MODULE_NAME_ACTION => 8, SERVICE_MODULE=>9, SERVICE_MODULE_ACTION=>10);
```

To illustrate how mapping priorities work, assume that in addition to the mapping of scott to the dev group, there is also a module name mapping as follows:

```
EXEC DBMS_RESOURCE_MANAGER.SET_CONSUMER_GROUP_MAPPING -
     (DBMS_RESOURCE_MANAGER.MODULE_NAME, 'backup', 'low priority');
```

In this example, the priority of the database user name is set to 4 (less important), while the priority of the module name is set to 3 (more important). Now if the session sets its module name to backup, presumably because it is going to perform a backup operation, then the session would be reassigned to the low priority consumer group, because that mapping has a higher priority.

To prevent unauthorized clients from setting their session attributes so that they map to higher priority consumer groups, user switch privileges for consumer

groups are enforced. This means that even though the attribute of a given session matches a mapping pair, the mapping is only considered valid if the session has the switching privilege for that particular consumer group. Sessions are automatically switched only to consumer groups for which they have been granted switch privileges.

Explicit Session Switching

A session can manually switch its own consumer group, given the privilege to do so, using the procedure `DBMS_SESSION.SWITCH_CURRENT_CONSUMER_GROUP`. Although there is no mapping for the `EXPLICIT` group setting, it is considered an attribute in the mapping priorities when deciding whether to use this `EXPLICIT` setting or use the consumer group mapping. In other words, if the `EXPLICIT` setting has the highest priority, then the session always switches consumer groups when it calls the `SWITCH_CURRENT_CONSUMER_GROUP` procedure.

On the other hand, if the `EXPLICIT` setting has the lowest priority, then the session only switches consumer groups if none of the other session attributes match in the mappings. Note that the Database Resource Manager considers a switch to have taken place even if the `SWITCH_CURRENT_COSUMER_GROUP` procedure is called to switch the session to the consumer group that it is already in.

Automatic Group Switching

Each session can be switched automatically to another consumer group via the mappings at distinct points in time:

- When the session first logs in, the mapping is evaluated to determine the initial group of the session.
- If the current mapping attribute of a session is A, then if the attribute A is set to a new value (only possible for runtime attributes) then the mapping is reevaluated and the session is switched to the appropriate consumer group.
- If a runtime session attribute is modified such that the current mapping becomes a different attribute B, then the session is switched.
- Whenever the client ID is set to a different value, the mapping is reevaluated and the session is switched.

Two things to note about the preceding rules are:

- If a runtime attribute for which a mapping is provided is set to the same value it already has, or if the client ID is set to the same value it already has, then no switching takes place.

- A session can be switched to the same consumer group it is already in. The effect of switching in this case is to zero out session statistics that are typically zeroed out during a switch to a different group (for example, the ACTIVE_TIME_IN_GROUP value of the session).

Enabling the Database Resource Manager

You enable the Database Resource Manager by setting the RESOURCE_MANAGER_PLAN initialization parameter. This parameter specifies the top plan, identifying the plan schema to be used for this instance. If no plan is specified with this parameter, the Database Resource Manager is not activated. The following example activates the Database Resource Manager and assigns the top plan as mydb_plan.

```
RESOURCE_MANAGER_PLAN = mydb_plan
```

You can also activate or deactivate the Database Resource Manager, or change the current top plan, using the ALTER SYSTEM statement. In this example, the top plan is specified as mydb_plan.

```
ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = 'mydb_plan';
```

An error message is returned if the specified plan does not exist in the data dictionary.

To deactivate the Database Resource Manager, issue the following statement:

```
ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = '';
```

When enabled, the DBMS Scheduler can automatically change the Resource Manager plan at Scheduler window boundaries. In some cases, this may be unacceptable. For example, if you have an important task to finish, and if you set the Resource Manager plan to give your task priority, then you expect that the plan will remain the same until you change it. However, because a Scheduler window could become activated after you have set your plan, the Resource Manager plan may change while your task is running.

To prevent this situation, you can set the RESOURCE_MANAGER_PLAN parameter to the name of the plan you want for the system and prepend the name with "FORCE:". Using the prefix FORCE indicates that the current Resource Manager plan can be changed only when the database administrator changes the value of the RESOURCE_MANAGER_PLAN parameter. This restriction can be lifted by reexecuting the command without prepending the plan name with "FORCE:".

```
ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = 'FORCE:mydb_plan';
```

Putting It All Together: Database Resource Manager Examples

This section provides some examples of resource plan schemas. The following examples are presented:

- Multilevel Schema Example
- Example of Using Several Resource Allocation Methods
- An Oracle-Supplied Plan

Multilevel Schema Example

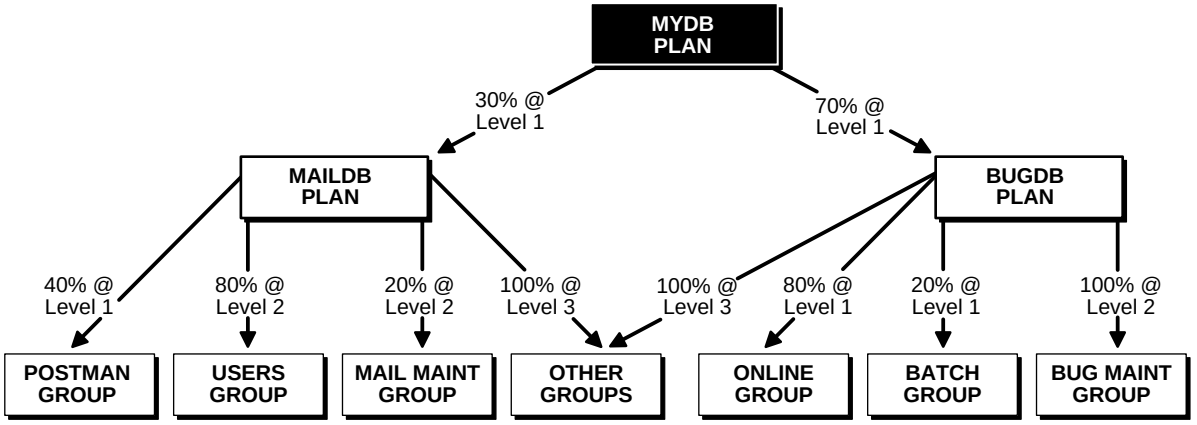
The following statements create a multilevel schema as illustrated in Figure 24–3. They use default plan and resource consumer group methods.

```
BEGIN
DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
DBMS_RESOURCE_MANAGER.CREATE_PLAN(PLAN => 'bugdb_plan',
   COMMENT => 'Resource plan/method for bug users sessions');
DBMS_RESOURCE_MANAGER.CREATE_PLAN(PLAN => 'maildb_plan',
   COMMENT => 'Resource plan/method for mail users sessions');
DBMS_RESOURCE_MANAGER.CREATE_PLAN(PLAN => 'mydb_plan',
   COMMENT => 'Resource plan/method for bug and mail users sessions');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Bug_Online_group',
   COMMENT => 'Resource consumer group/method for online bug users sessions');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Bug_Batch_group',
   COMMENT => 'Resource consumer group/method for batch job bug users sessions');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Bug_Maintenance_group',
   COMMENT => 'Resource consumer group/method for users sessions for bug db maint');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Mail_users_group',
   COMMENT => 'Resource consumer group/method for mail users sessions');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Mail_Postman_group',
   COMMENT => 'Resource consumer group/method for mail postman');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Mail_Maintenance_group',
   COMMENT => 'Resource consumer group/method for users sessions for mail db maint');
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'bugdb_plan',
   GROUP_OR_SUBPLAN => 'Bug_Online_group',
   COMMENT => 'online bug users sessions at level 1', CPU_P1 => 80, CPU_P2=> 0,
   PARALLEL_DEGREE_LIMIT_P1 => 8);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'bugdb_plan',
   GROUP_OR_SUBPLAN => 'Bug_Batch_group',
   COMMENT => 'batch bug users sessions at level 1', CPU_P1 => 20, CPU_P2 => 0,
   PARALLEL_DEGREE_LIMIT_P1 => 2);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'bugdb_plan',
   GROUP_OR_SUBPLAN => 'Bug_Maintenance_group',
   COMMENT => 'bug maintenance users sessions at level 2', CPU_P1 => 0, CPU_P2 => 100,
   PARALLEL_DEGREE_LIMIT_P1 => 3);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'bugdb_plan',
```

```
   GROUP_OR_SUBPLAN => 'OTHER_GROUPS',
   COMMENT => 'all other users sessions at level 3', CPU_P1 => 0, CPU_P2 => 0,
   CPU_P3 => 100);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'maildb_plan',
   GROUP_OR_SUBPLAN => 'Mail_Postman_group',
   COMMENT => 'mail postman at level 1', CPU_P1 => 40, CPU_P2 => 0,
   PARALLEL_DEGREE_LIMIT_P1 => 4);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'maildb_plan',
   GROUP_OR_SUBPLAN => 'Mail_users_group',
   COMMENT => 'mail users sessions at level 2', CPU_P1 => 0, CPU_P2 => 80,
   PARALLEL_DEGREE_LIMIT_P1 => 4);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'maildb_plan',
   GROUP_OR_SUBPLAN => 'Mail_Maintenance_group',
   COMMENT => 'mail maintenance users sessions at level 2', CPU_P1 => 0, CPU_P2 => 20,
   PARALLEL_DEGREE_LIMIT_P1 => 2);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'maildb_plan',
   GROUP_OR_SUBPLAN => 'OTHER_GROUPS',
   COMMENT => 'all other users sessions at level 3', CPU_P1 => 0, CPU_P2 => 0,
   CPU_P3 => 100);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'mydb_plan',
   GROUP_OR_SUBPLAN => 'maildb_plan',
   COMMENT=> 'all mail users sessions at level 1', CPU_P1 => 30);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'mydb_plan',
   GROUP_OR_SUBPLAN => 'bugdb_plan',
   COMMENT => 'all bug users sessions at level 1', CPU_P1 => 70);
DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
END;
```

The preceding call to VALIDATE_PENDING_AREA is optional because the validation is implicitly performed in SUBMIT_PENDING_AREA.

Figure 24–3 Multilevel Schema



Example of Using Several Resource Allocation Methods

The example presented here could represent a plan for a database supporting a packaged ERP (Enterprise Resource Planning) or CRM (Customer Relationship Management). The work in such an environment can be highly varied. There may be a mix of short transactions and quick queries, in combination with longer running batch jobs that include large parallel queries. The goal is to give good response time to OLTP (Online Transaction Processing), while allowing batch jobs to run in parallel.

The plan is summarized in the following table.

Group	CPU Resource Allocation %	Active Session Pool Parameters	Automatic Switching Parameters	Maximum Estimated Execution Time	Undo Pool
`oltp`	Level 1: 80%		Switch to group: `batch` Switch time: 3 Use estimate: `TRUE`	--	Size: 200K
`batch`	Level 2: 100%	Pool size: 5 Timeout: 600	--	Time: 3600	--
`OTHER_GROUPS`	Level 3: 100%	--	--	--	--

The following statements create the preceding plan, which is named `erp_plan`:

```
BEGIN
DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
DBMS_RESOURCE_MANAGER.CREATE_PLAN(PLAN => 'erp_plan',
  COMMENT => 'Resource plan/method for ERP Database');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'oltp',
  COMMENT => 'Resource consumer group/method for OLTP jobs');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'batch',
  COMMENT => 'Resource consumer group/method for BATCH jobs');
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'erp_plan',
  GROUP_OR_SUBPLAN => 'oltp', COMMENT => 'OLTP sessions', CPU_P1 => 80,
  SWITCH_GROUP => 'batch', SWITCH_TIME => 3,SWITCH_ESTIMATE => TRUE,
  UNDO_POOL => 200);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'erp_plan',
  GROUP_OR_SUBPLAN => 'batch', COMMENT => 'BATCH sessions', CPU_P2 => 100,
  ACTIVE_SESS_POOL_P1 => 5, QUEUEING_P1 => 600,
  MAX_EST_EXEC_TIME => 3600);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'erp_plan',
  GROUP_OR_SUBPLAN => 'OTHER_GROUPS', COMMENT => 'mandatory', CPU_P3 => 100);
DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
END;
```

An Oracle-Supplied Plan

Oracle Database provides one default resource manager plan, SYSTEM_PLAN, which gives priority to system sessions. SYSTEM_PLAN is defined as follows:

Resource Consumer Group	CPU Resource Allocation		
	Level 1	Level 2	Level 3
SYS_GROUP	100%	0%	0%
OTHER_GROUPS	0%	100%	0%
LOW_GROUP	0%	0%	100%

The database-provided groups in this plan are:

- SYS_GROUP is the initial consumer group for the users SYS and SYSTEM.
- OTHER_GROUPS applies collectively to all sessions that belong to a consumer group that is not part of the currently active plan schema.
- LOW_GROUP provides a group having lower priority than SYS_GROUP and OTHER_GROUPS in this plan. It is up to you to decide which user sessions will be part of LOW_GROUP. Switch privilege is granted to PUBLIC for this group.

These groups can be used, or not used, and can be modified or deleted.

You can use this simple database-supplied plan if it is appropriate for your environment.

Monitoring and Tuning the Database Resource Manager

To effectively monitor and tune the Database Resource Manager, you must design a representative environment. The Database Resource Manager works best in large production environments in which system utilization is high. If a test places insufficient load on the system, measured CPU allocations can be very different from the allocations specified in the active resource plan. This is because the Database Resource Manager does not attempt to enforce CPU allocation percentage limits as long as consumer groups are getting the resources they need.

Creating the Environment

To create a representative environment, there must be sufficient load (demand for CPU resources) to make CPU resources scarce. If the following rules are followed, the test environment should generate actual (measured) resource allocations that match those specified in the active resource plan.

1. Create the minimum number of concurrently running processes required to generate sufficient load. This is the larger of:
   - Four processes for each consumer group
   - 1.5 * (number of processors) for each consumer group. If the result is not an integer, round up.
2. Each and every process must be capable of consuming all of the CPU resources allocated to the consumer group in which it runs. Write resource intensive programs that continue to spin no matter what happens. This can be as simple as:

```
BEGIN
DECLARE
    m NUMBER;
  BEGIN
    FOR i IN 1..100000 LOOP
      FOR j IN 1..100000 LOOP
        m := sqrt(4567);
      END LOOP;
    END LOOP;
  END;
```

```
END;
/
```

Why Is This Necessary to Produce Expected Results?

When every group can secure as much CPU resources as it demands, the Database Resource Manager first seeks to maximize system throughput, not to enforce allocation percentages. For example, consider the following conditions:

- There is only one process available to run for each consumer group.
- Each process runs continuously.
- There are four CPUs.

In this case, the measured CPU allocation to each consumer group will be 25%, no matter what the allocations specified in the active resource plan.

Another factor determines the calculation in (1) in the preceding section. Processor affinity scheduling at the operating system level can distort CPU allocation on underutilized systems. This is explained in the following paragraphs.

Until the number of concurrently running processes reaches a certain level, typical operating system scheduling algorithms will prevent full utilization. The Database Resource Manager controls CPU usage by restricting the number of running processes. By deciding which processes are allowed to run and for what duration, the Database Resource Manager controls CPU resource allocation. When a CPU has resources available, and other processors are fully utilized, the operating system migrates processes to the underutilized processor, but not immediately.

With processor affinity, the operating system waits (for a time) to migrate processes, "hoping" that another process will be dispatched to run instead of forcing process migration from one CPU to another. On a fully loaded system with enough processes waiting, this strategy will work. In large production environments, processor affinity increases performance significantly, because invalidating the current CPU cache and then loading the new one is quite expensive. Since processes have processor affinity on most platforms, more processes than CPUs for each consumer group must be run. Otherwise, full system utilization is not possible.

Monitoring Results

Use the `V$RSRC_CONSUMER_GROUP` view to monitor CPU usage. It provides the cumulative amount of CPU time consumed by all sessions in each consumer group. It also provides a number of other measures helpful for tuning.

```
SQL> SELECT NAME, CONSUMED_CPU_TIME FROM V$RSRC_CONSUMER_GROUP;

NAME                             CONSUMED_CPU_TIME
-------------------------------- -----------------
OTHER_GROUPS                                14301
TEST_GROUP                                   8802
TEST_GROUP2                                     0

3 rows selected.
```

Interaction with Operating-System Resource Control

The Oracle Database server expects a static configuration and allocates internal resources, such as latches, from available resources detected at database startup. The database might not perform optimally and can become unstable if resource configuration changes very frequently.

Guidelines for Using Operating-System Resource Control

If you do choose to use Operating-system resource control with Oracle Database, then it should be used judiciously, according to the following guidelines:

1. Operating-system resource control should not be used concurrently with the Database Resource Manager, because neither of them are aware of each other's existence. As a result, both the operating system and Database Resource Manager try to control resource allocation in a manner that causes unpredictable behavior and instability of Oracle Database.

   - If you want to control resource distribution within an instance, use the Database Resource Manager and turn off operating-system resource control.
   - If you have multiple instances on a node and you want to distribute resources among them, use operating-system resource control, not the Database Resource Manager.

   Note: Oracle Database currently does not support the use of both tools simultaneously. Future releases might allow for their interaction on a limited scale.

2. In an Oracle Database environment, the use of an operating-system resource manager, such as Hewlett Packard's Process Resource Manager (PRM) or Sun's

Solaris Resource Manager (SRM), is supported only if all of the following conditions are met:

- Each instance must be assigned to a dedicated operating-system resource manager group or managed entity.
- The dedicated entity running all the instance's processes must run at one priority (or resource consumption) level.
- Process priority management must not be enabled.

Caution: Management of individual database processes at different priority levels (for example, using the `nice` command on UNIX platforms) is not supported. Severe consequences, including instance crashes, can result. You can expect similar undesirable results if operating-system resource control is permitted to manage memory on which an Oracle Database instance is pinned.

---

3. If you chose to use operating-system resource control, make sure you turn off the Database Resource Manager. By default, the Database Resource Manager is turned off. If it is not, you can turn it off by issuing the following statement:

```
ALTER SYSTEM SET RESOURCE_MANAGER_PLAN='';
```

Also remember to reset this parameter in your initialization parameter file, so that the Database Resource Manager is not activated the next time the database is started up.

Dynamic Reconfiguration

Tools such as Sun's processor sets and dynamic system domains work well with an Oracle Database. There is no need to restart an instance if the number of CPUs changes.

The database dynamically detects any change in the number of available CPUs and reallocates internal resources. On most platforms, the database automatically adjusts the value of the `CPU_COUNT` initialization parameter to the number of available CPUs.

Viewing Database Resource Manager Information

The following table lists views that are associated with Database Resource Manager:

View	Description
DBA_RSRC_CONSUMER_GROUP_PRIVS USER_RSRC_CONSUMER_GROUP_PRIVS	DBA view lists all resource consumer groups and the users and roles to which they have been granted. USER view lists all resource consumer groups granted to the user.
DBA_RSRC_CONSUMER_GROUPS	Lists all resource consumer groups that exist in the database.
DBA_RSRC_MANAGER_SYSTEM_PRIVS USER_RSRC_MANAGER_SYSTEM_PRIVS	DBA view lists all users and roles that have been granted Database Resource Manager system privileges. USER view lists all the users that are granted system privileges for the DBMS_RESOURCE_MANAGER package.
DBA_RSRC_PLAN_DIRECTIVES	Lists all resource plan directives that exist in the database.
DBA_RSRC_PLANS	Lists all resource plans that exist in the database.
DBA_RSRC_GROUP_MAPPINGS	Lists all of the various mapping pairs for all of the session attributes
DBA_RSRC_MAPPING_PRIORITY	Lists the current mapping priority of each attribute
DBA_USERS USERS_USERS	DBA view contains information about all users of the database. Specifically, for the Database Resource Manager, it contains the initial resource consumer group for the user. USER view contains information about the current user, and specifically, for the Database Resource Manager, it contains the current user's initial resource consumer group.
V$ACTIVE_SESS_POOL_MTH	Displays all available active session pool resource allocation methods.
V$PARALLEL_DEGREE_LIMIT_MTH	Displays all available parallel degree limit resource allocation methods.
V$QUEUEING	Displays all available queuing resource allocation methods.
V$RSRC_CONSUMER_GROUP	Displays information about active resource consumer groups. This view can be used for tuning.
V$RSRC_CONSUMER_GROUP_CPU_MTH	Displays all available CPU resource allocation methods for resource consumer groups.
V$RSRC_PLAN	Displays the names of all currently active resource plans.
V$RSRC_PLAN_CPU_MTH	Displays all available CPU resource allocation methods for resource plans.
V$SESSION	Lists session information for each current session. Specifically, lists the name of the resource consumer group of each current session.

You can use these views for viewing privileges, viewing plan schemas, or you can monitor them to gather information for tuning the Database Resource Manager. Some examples of their use follow.

> **See Also:** *Oracle Database Reference* for detailed information about the contents of each of these views

Viewing Consumer Groups Granted to Users or Roles

The DBA_RSRC_CONSUMER_GROUP_PRIVS view displays the consumer groups granted to users or roles. Specifically, it displays the groups to which a user or role is allowed to belong or be switched. For example, in the view shown below, user `scott` can belong to the consumer groups `market` or `sales`, he has the ability to assign (grant) other users to the `sales` group but not the `market` group. Neither group is his initial consumer group.

```
SQL> SELECT * FROM DBA_RSRC_CONSUMER_GROUP_PRIVS;

GRANTEE                        GRANTED_GROUP                  GRA INI
------------------------------ ------------------------------ --- ---
PUBLIC                         DEFAULT_CONSUMER_GROUP         YES YES
PUBLIC                         LOW_GROUP                      NO  NO
SCOTT                          MARKET                         NO  NO
SCOTT                          SALES                          YES NO
SYSTEM                         SYS_GROUP                      NO  YES
```

Scott was granted the ability to switch to these groups using the DBMS_RESOURCE_MANAGER_PRIVS package.

Viewing Plan Schema Information

This example shows using the DBA_RSRC_PLANS view to display all of the resource plans defined in the database. All of the plans displayed are active, meaning they are not staged in the pending area

```
SQL> SELECT PLAN,COMMENTS,STATUS FROM DBA_RSRC_PLANS;

PLAN         COMMENTS                                                 STATUS
-----------  -------------------------------------------------------  ------
SYSTEM_PLAN  Plan to give system sessions priority                    ACTIVE
BUGDB_PLAN   Resource plan/method for bug users sessions             ACTIVE
MAILDB_PLAN  Resource plan/method for mail users sessions            ACTIVE
MYDB_PLAN    Resource plan/method for bug and mail users sessions    ACTIVE
GREAT_BREAD  Great plan for great bread                              ACTIVE
```

```
ERP_PLAN     Resource plan/method for ERP Database                    ACTIVE

6 rows selected.
```

Viewing Current Consumer Groups for Sessions

You can use the V$SESSION view to display the consumer groups that are currently assigned to sessions.

```
SQL> SELECT SID,SERIAL#,USERNAME,RESOURCE_CONSUMER_GROUP FROM V$SESSION;

SID    SERIAL#  USERNAME                  RESOURCE_CONSUMER_GROUP
-----  -------  ------------------------  --------------------------------
.
.
.
   11       136 SYS                       SYS_GROUP
   13     16570 SCOTT                     SALES

10 rows selected.
```

Viewing the Currently Active Plans

This example sets mydb_plan, as created by the statements shown earlier in "Multilevel Schema Example" on page 24-32, as the top level plan. The V$RSRC_PLAN view is queried to display the currently active plans.

```
SQL> ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = mydb_plan;

System altered.

SQL> SELECT NAME, IS_TOP_PLAN FROM V$RSRC_PLAN;

NAME                             IS_TO
-------------------------------------
MYDB_PLAN                        TRUE
MAILDB_PLAN                      FALSE
BUGDB_PLAN                       FALSE
```

25

Moving from DBMS_JOB to DBMS_SCHEDULER

Oracle Database provides advanced scheduling capabilities through the database Scheduler, a collection of functions and procedures in the DBMS_SCHEDULER package. The Scheduler offers far more functionality than the DBMS_JOB package, which was the previous Oracle Database job scheduler. This chapter discusses briefly how you can take statements created with DBMS_JOB and rewrite them using DBMS_SCHEDULER.

> **See Also:** The documentation on supplied PL/SQL packages that came with your Oracle9i software for information on the DBMS_JOB package

This chapter contains the following topic:

- Moving from DBMS_JOB to DBMS_SCHEDULER

Moving from DBMS_JOB to DBMS_SCHEDULER

This section illustrates some examples of how you can take jobs created with the DBMS_JOB package and rewrite them using the DBMS_SCHEDULER package.

Creating a Job

An example of creating a job using DBMS_JOB is the following:

```
VARIABLE jobno NUMBER;
BEGIN
DBMS_JOB.SUBMIT(:jobno, 'INSERT INTO employees VALUES (7935, ''SALLY'',
   ''DOGAN'', ''sally.dogan@xyzcorp.com'', NULL, SYSDATE, ''AD_PRES'', NULL,
```

```
    NULL, NULL, NULL);', SYSDATE, 'SYSDATE+1');
COMMIT;
END;
/
```

An equivalent statement using DBMS_SCHEDULER is the following:

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB(
   job_name          =>  'job1',
   job_type          =>  'PLSQL_BLOCK',
   job_action        =>  'INSERT INTO employees VALUES (7935, ''SALLY'',
     ''DOGAN'', ''sally.dogan@xyzcorp.com'', NULL, SYSDATE,''AD_PRES'', NULL,
      NULL, NULL, NULL);');
   start_date        =>  SYSDATE,
   repeat_interval   =>  'FREQ = DAILY; INTERVAL = 1');
END;
/
```

Altering a Job

An example of altering a job using DBMS_JOB is the following:

```
BEGIN
DBMS_JOB.WHAT(31, 'INSERT INTO employees VALUES (7935, ''TOM'', ''DOGAN'',
   ''tom.dogan@xyzcorp.com'', NULL, SYSDATE,''AD_PRES'', NULL,
   NULL, NULL, NULL);');
COMMIT;
END;
/
```

This changes the action for job 31 to insert a different value. An equivalent statement using DBMS_SCHEDULER is the following:

```
BEGIN
DBMS_SCHEDULER.SET_ATTRIBUTE(
   name          => 'JOB1',
   attribute     => 'job_action',
   value         => 'INSERT INTO employees VALUES (7935, ''TOM'', ''DOGAN'',
      ''tom.dogan@xyzcorp.com'', NULL, SYSDATE, ''AD_PRES'', NULL,
      NULL, NULL, NULL);');
END;
/
```

Removing a Job from the Job Queue

The following example removes a job using DBMS_JOB, where 14144 is the number of the job being run:

```
BEGIN
DBMS_JOB.REMOVE(14144);
COMMIT;
END;
/
```

Using DBMS_SCHEDULER, you would issue the following statement instead:

```
BEGIN
   DBMS_SCHEDULER.DROP_JOB('myjob1');
END;
/
```

See Also:

- *PL/SQL Packages and Types Reference* for more information about the DBMS_SCHEDULER package
- Chapter 26, "Overview of Scheduler Concepts"
- Chapter 28, "Administering the Scheduler"

26

Overview of Scheduler Concepts

Oracle provides advanced scheduling capabilities through the database Scheduler. This chapter introduces you to its concepts, and discusses the following topics:

- Overview of the Scheduler
- Basic Scheduler Concepts
- Advanced Scheduler Concepts
- Scheduler Architecture

> **Note:** This chapter discusses the use of the Oracle-supplied `DBMS_SCHEDULER` package to administer scheduling capabilities. You can also use Oracle Enterprise Manager (EM) as an easy-to-use graphical interface for many of the same capabilities.
>
> See the *PL/SQL Packages and Types Reference* for `DBMS_SCHEDULER` syntax and the Oracle Enterprise Manager documentation set for more information regarding EM.

Overview of the Scheduler

Organizations have too many tasks and manually dealing with each one can be daunting. To help you simplify these management tasks, as well as offering a rich set of functionality for complex scheduling needs, Oracle provides a collection of functions and procedures in the `DBMS_SCHEDULER` package. Collectively, these functions are called the Scheduler, and they are callable from any PL/SQL program.

The Scheduler enables database administrators and application developers to control when and where various tasks take place in the database environment. These tasks can be time consuming and complicated, so using the Scheduler can

help you to improve the management and planning of these tasks. In addition, by ensuring that many routine database tasks occur without manual intervention, you can lower operating costs, implement more reliable routines, minimize human error, and shorten the time windows needed.

Some typical examples of using the Scheduler are:

- Database administrators can schedule and monitor database maintenance jobs such as backups or nightly data warehousing loads and extracts.
- Application developers can create programs and program libraries that end users can use to create or monitor their own jobs. In addition to typical database jobs, you can schedule and monitor jobs that run as part of an application suite.

What Can the Scheduler Do?

The Scheduler provides complex enterprise scheduling functionality, which you can use to:

- Schedule job execution based on time

  The most basic capability of a job scheduler is the ability to schedule a job to run at a particular date and time. The Scheduler enables you to reduce your operating costs by enabling you to schedule execution of jobs. For example, consider the situation where a patch needs to be applied to a database that is in production. To minimize disruptions, this task will need to be performed during non-peak hours. This can be easily accomplished using the Scheduler. Instead of having IT personnel manually carry out this task during non-peak hours, you can instead create a job and schedule it to run at a specified time using the Scheduler. See "Creating Jobs" on page 27-3 for more information.

- Reuse existing programs and schedules

  A program is a collection of metadata about what will be run by the Scheduler. A schedule specifies when the program will be executed. Programs and schedules are separate entities and are stored in the database as independent database objects making them available for reuse.

  You can create a job using existing programs and schedules. Parameter values specified when creating the job will override the default values specified in the program, thus enabling other users to customize the program to meet their needs. This approach enables different users to create different jobs using existing programs and schedules without having to redefine them each time. For example, analyzing tables is a common task that most database administrators perform on a regular basis. Because the metadata of what the

Scheduler needs to run is the same in all cases, you can create one program that takes the table name as a parameter to analyze tables. This way, only the table name is different. By specifying the table name when creating each job, the same program can be reused by different jobs, thus eliminating the need to redefine the program each time. Different jobs can also schedule the same program to run at different times by using different schedules. See "Creating Programs" on page 27-13 for more information.

- Schedule job processing in a way that models your business requirements

  The Scheduler enables limited computing resources to be allocated appropriately among competing jobs, thus aligning job processing with your business needs. This is accomplished on two levels:

  – Jobs that share common characteristic and behavior can be grouped into larger entities called job classes. You can prioritize among the classes by controlling the resources allocated to each class. This enables you to ensure that your critical jobs have priority and have enough resources to complete. For example, if you have a critical project to load a data warehouse, then you can combine all the data warehousing jobs into one class and give priority to it over other jobs by allocating it a high percentage of the available resources.

  – The Scheduler takes prioritization among of jobs one step further, by providing you the ability to change the prioritization based on a schedule. Because your definition of a critical job can change over time, the Scheduler enables you to also change the priority among your jobs over that time frame. For example, you may consider the jobs to load a data warehouse to be critical jobs during non-peak hours but not during peak hours. In such a case, you can change the priority among the classes by changing the resource allocated to each class. See "Creating Job Classes" on page 27-24 and "Creating Windows" on page 27-27 for more information.

- Manage and monitor jobs

  There are multiple states that a job undergoes from its creation to its completion. Scheduler activity is logged and information such as the status of the job and the time to completion of the job can be easily tracked. This information is stored in views and can be easily queried using Enterprise Manager or a SQL query. These views provide valuable information about jobs and their execution that can help you schedule and manage your jobs better. For example, a DBA can easily track all jobs that failed for user `scott`. See "Monitoring and Managing the Scheduler" on page 28-7.

- Execute and manage jobs in a clustered environment

A cluster is a set of database instances that cooperates to perform the same task. Oracle Real Application Clusters (RAC) provides scalability and reliability without any change to your applications. The Scheduler fully supports execution of jobs in such a clustered environment. To balance the load on your system and for better performance, you can also specify the service where you want a job to run. See "Using the Scheduler in RAC Environments" on page 26-14 for more information.

Basic Scheduler Concepts

The Scheduler offers a modularized approach for managing tasks within the Oracle environment. By breaking down a task into its components such as time, location, and database object, the Scheduler offers you an easier way to manage your database environment. Another advantage of this modularization is that different users can perform similar tasks with only small modifications.

In the Scheduler, all the components are database objects like a table, which enables you to use normal Oracle privileges.

Oracle provides transient jobs, which are deleted once executed. This means that all the metadata about the job is deleted from the database. Of course, the job log will contain an entry for the transient job so it can be audited. In addition to transient jobs, the Scheduler also enables you to create persistent jobs.

General Rules for all Database Objects

The following rules apply for all objects when using the Scheduler:

- If you try to drop an object that does not exist, a PL/SQL exception is raised saying that the object does not exist.
- If you try to enable or disable an object that is already enabled or disabled, no error is generated.
- If you try to alter an object that does not exist, an exception is raised saying that the object does not exist.

The basic elements of the Scheduler are:

- Programs
- Schedules
- Jobs

Programs

A program is a collection of metadata about what will be run by the Scheduler. It includes information such as the name of the program, program action (for example, a procedure or executable name), program type (for example, PL/SQL and Java stored procedures or PL/SQL anonymous blocks) and the number of arguments required for the program.

A program is a separate entity from a job. Jobs can be created using existing programs, which means that different jobs can use the same program. Given the right privileges, different users can use the same program without having to redefine it. This enables the creation of program libraries, where users can select from a list of existing programs.

See "Creating Programs" on page 27-13 for more information.

Schedules

A schedule specifies when and how many times a job is executed. Jobs can be scheduled for processing at a later time or immediately. For jobs to be executed at a later time, the user can specify a date and time when the job should start. For jobs that repeat over a period of time, an end date and time can be specified, which indicates when the schedule expires. A schedule also has a limit associated with it, which is expressed as a time duration. The schedule limit indicates that a job should not be run if the duration has elapsed after the scheduled start time and the job has not yet been started. If the job was a repeating job, it is rescheduled.

Similar to programs, schedules are database entities and can be saved in the database. Users with the right privileges can share saved schedules. For example, the end of quarter may be a common schedule for many jobs. Instead of each user having to redefine the schedule each time, they can all point to a saved schedule.

Some examples of schedules you might use to control time-based jobs are:

- Run on Wednesday, December 26th, 2001 at 2pm
- Run every Monday, at 8am, starting on December 26th, 2001, and ending on January 31st, 2002
- Run on every working day

See "Creating Schedules" on page 27-18 for more information.

Jobs

A job is a user-defined task that is scheduled to run one or more times. It is a combination of what (the action) needs to be executed and when (the schedule). Users with the right privileges can create jobs either by simply specifying the action and schedule at the time of job creation, or by using existing programs and schedules. To customize an existing program, users can specify values for the program arguments during job creation, which will override the defaults that were specified when the program was created.

A common example of a job is backing up data on a personal computer. This is a task that most users perform on a nightly basis. It would therefore be best to create a program for this task that can be shared among different users. The program action would be the backup script and the program would have one argument, the path where the files that need to be backed up reside. Each user would create a job that points to this program and specify the path that is unique to them during the job creation, thus customizing the program for their own needs.

See "Creating Jobs" on page 27-3 for more information.

Job Instances

A job instance represents a specific run of a job. Jobs that are scheduled to run only once will have only one instance, however, jobs that have a repeating schedule will have multiple instances, with each run of the job representing an instance. For example, a job that is scheduled to run on Tuesday, Oct. 8th 2002 will have one instance. A job that runs daily at noon for a week will have seven instances, one for each time the job ran. So, in this case, the run on Tuesday at noon represents one instance of the job.

When a job is created, only one entry is added to the job table to represent the job, however, each time the job runs, an entry will be added to the job log. Therefore, if you create a job that has a repeating schedule, you should find one entry in the job table and multiple entries in the job log. Each run of the job is assigned an instance ID that can be used to monitor job progress and manage future runs. They provide specific information about a particular run, for example, the job status and the start and end time.

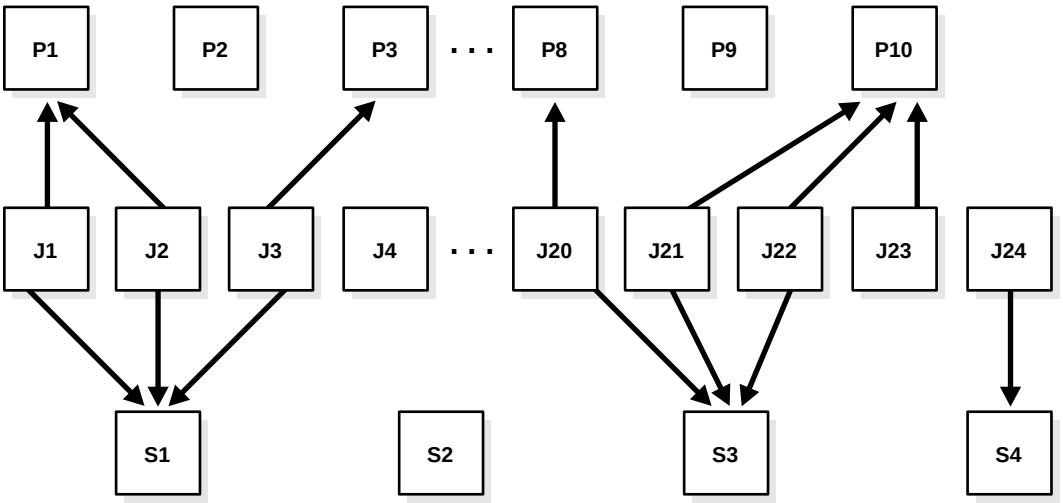
See "How to View Scheduler Information" on page 28-8 for more information.

How Programs, Jobs, and Schedules are Related

To determine when, where, and what will be executed in the database, you need to assign relationships among programs, jobs, and schedules. You can, however, leave

a program, job, or schedule unassigned if you wish. You could, for example, create a job that calculates a year end inventory and leave it unassigned. That could be J4 in Figure 26–1. Similarly, schedules and programs could exist for year end tasks and be left unassigned. Figure 26–1 illustrates the relationship among programs, jobs, and schedules.

Figure 26–1 Relationships Among Programs, Jobs, and Schedules



To understand Figure 26–1, consider a situation where tables are being analyzed. In this example, P1 would be a program to analyze a table using the DBMS_STATS package. The program takes in a variable for the table name. Two jobs, J1 and J2, both point to the same program, which specifies a different table name. Finally, schedule S1 could have a value of 2AM for when the jobs should be performed. The end result is that the two tables named in J1 and J2 would be analyzed at 2AM.

Note that P2, P9, J4, and S2 are entities that have nothing pointed to them in Figure 26–1, so, for example, J4 could be self-contained with all relevant information inside the job itself. Also note that more than one job can map to a program or schedule.

See "Examples of Using the Scheduler" on page 28-23 for more examples.

Advanced Scheduler Concepts

Many Scheduler capabilities enable database administrators to control more advanced aspects of scheduling. Typically, these topics are not as important for application developers.

This section discusses the following advanced topics:

- Job Classes
- Windows
- Window Groups

Job Classes

A job class is a way of grouping jobs into larger entities, thus enabling you to prioritize access to the job slaves among those job classes. This means that you can, for example, make sure a CEO's request will begin before a routine job. In addition to using job classes for resource allocation, you can also use them for setting job characteristics or behavior that must be the same for all the jobs in the class.

When defining job classes, you should try to classify jobs by functionality. Consider dividing jobs into groups that access similar data, such as marketing, production, sales, finance, and human resources.

Within a job class, you can:

- Specify attributes at the class level. For example, you can specify the same policy for purging log entries of all payroll jobs.
- Specify the order in which a job is started. You can assign priority values of 1-5 to individual jobs so that jobs with a higher relative priority start before jobs with a lower priority. If two jobs have the same assigned value, the first job enqueued takes precedence.
- Ensure that a job is started only if there are no runnable jobs with a higher priority in that class. In the case of resource contention, this ensures that you do not have a less important job preventing the timely completion of a more important one.

Some of the restrictions to keep in mind are:

- A job must be part of exactly one class. When a job is created, you must specify which class the job is part of. If you do not specify which class the job is part of, it automatically becomes part of the default class.

- Dropping a class while there are still jobs in that class results in an error. You can force a class to be dropped even if there are still jobs that are members of that class, but all jobs referring to that class will automatically be disabled. In this case, the class for these jobs will be reset to the default system class. Jobs belonging to the dropped class that are already running will continue to run under class settings determined at the start of the job.

Note: Job priorities are only used to prioritize among jobs in the same class.

There is no guarantee that a high priority job in class A will be started before a low priority job in class B, even if they share the same schedule. Prioritizing among jobs of different classes is purely done on a class resource allocation basis.

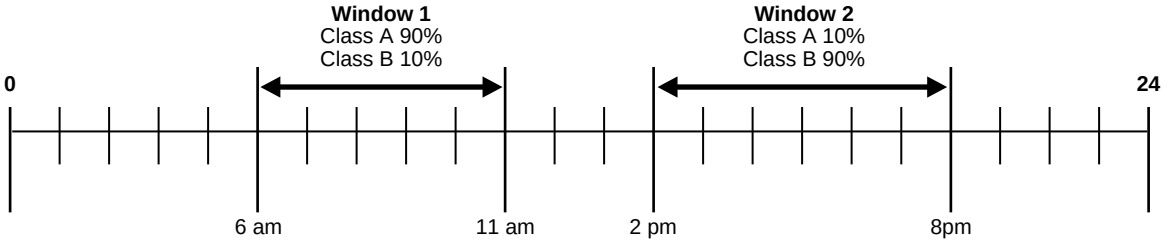
Windows

A window enables you to change resource allocation during a time period such as time of day or time of the sales year. You do this so you can control which groups of users have what level of priority. For example, you could control access to a printer so product managers have a lower priority than executives. A window is represented by an interval of time with a well-defined beginning and end, such as "from 12am-6am".

You can assign a priority with each window. If windows overlap, the window with the highest priority is chosen over other windows with lower priorities. See "Scheduler Architecture" on page 26-12 for details.

Figure 26–2 illustrates a simple scenario where separate windows have been assigned their own priorities. In Figure 26–2, Class A represents users with more need for resources in the morning while Class B represents users with more need for resources in the afternoon.

Figure 26–2 Windows



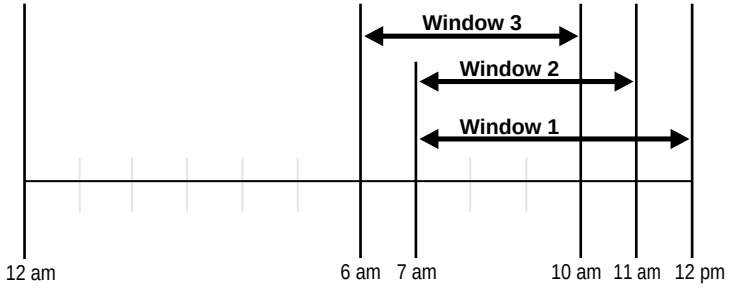
See "Creating Windows" on page 27-27 for examples of creating windows.

Overlapping Windows

You use windows to guarantee that resources will be available when you need them. Therefore, you should try to avoid having more than one window active at the same time to minimize the potential for resources going to a less important task. It is not an error, however, for windows to overlap.

Figure 26–3 illustrates a simple case of windows overlapping.

Figure 26–3 Overlapping Windows



There are three windows in Figure 26–3, so some rules need to be assigned for choosing which window will take precedence. The order of precedence for windows is as follows:

1. A high priority is chosen over a low priority. Oracle recommends that you choose low as the normal setting so you can guarantee resources for important high priority jobs.
2. A window gets to finish before the Scheduler switches to another window.

3. When a window finishes, the window with the highest percentage of its duration remaining is chosen.

If at the end of a window there is still overlap with one or more windows, the Scheduler tries to switch to the window that has the highest priority and then, if the windows have the same priorities, to the one with the highest percentage of its duration remaining.

Overlapping windows are handled differently based on whether they have the same priority or not. When two or more windows of the same priority overlap, Oracle continues to run the window that was already running. After this window finishes running, Oracle switches to the one that still has the highest percentage of its duration left.

When high and low priority windows overlap, the Scheduler switches to the high priority window if it is currently running a low priority window. Jobs currently running that had a schedule naming the low priority window may be stopped depending on the behavior you assigned when you created the job. In addition, whenever two windows overlap, an entry is written in the Scheduler log.

Window Groups

You can group jobs for ease of use. An example would be to combine weekends, times from 12AM to 7AM, and holidays into something called "downtime". This downtime window group offers you more control over when jobs are run. In this case, perhaps the data warehousing department would only want their queries run during this time when they might be assigned a high percentage of available resources.

A window group is only to combine windows and is only for jobs that use a window as a schedule.

Example of Window Group

A typical example of a window group is to have windows for weeknight, weekend, and holidays. A window group could be for combining them.

- `window1`: for weeknights
- `window2`: for weekends
- `mywindowgroup`(`window1`, `window2`): for a combination of weeknights and weekends

See "Creating Window Groups" on page 27-38 for examples of creating window groups.

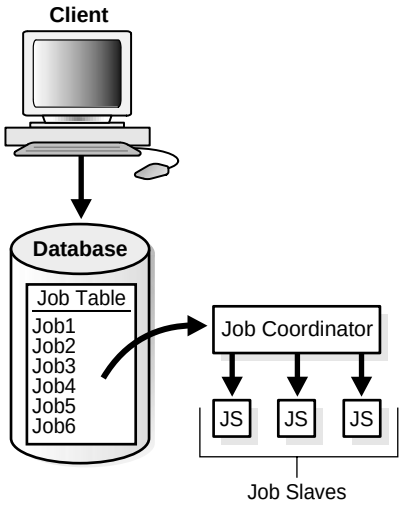
Scheduler Architecture

This section discusses the Scheduler's architecture, and describes:

- The Job Table
- The Job Coordinator
- How Jobs Execute
- Job Slaves
- Using the Scheduler in RAC Environments

Figure 26–4 illustrates how jobs are handled by the database.

Figure 26–4 Scheduler Components



The Job Table

The job table is a container for all the jobs, with one table per database. The job table stores information for all jobs such as the owner name or the level of logging. You can find this information in the `*_SCHEDULER_JOBS` views.

Jobs are database objects, and can therefore accumulate and take up too much space. To avoid this, a type of job called `transient` is available and is the default. Jobs created as `transient` are deleted by a background process after the jobs are

finished. You can, of course, choose to use the `persistent` type of job, which allows for jobs to be kept for additional logging purposes.

See "How to View Scheduler Information" on page 28-8 for the available job views and administration.

The Job Coordinator

The job coordinator is a background process (cjqNNN) that is automatically started when jobs must be run, or windows must be opened. It is automatically brought down after a sustained period of Scheduler inactivity. The job coordinator:

- Controls and spawns the job slaves
- Queries the job table
- Picks up jobs from the job table on a regular basis and places them in a memory cache. This improves performance by avoiding going to the disk
- Takes jobs from the memory cache and passes them to job slaves for execution
- Cleans up the job slave pool when slaves are no longer needed
- Goes to sleep when no jobs are scheduled
- Wakes up when a new job is about to be executed or a job was created using the `CREATE_JOB` procedure

You do not need to set when the job coordinator checks the job table; the system chooses the time frame automatically.

One job coordinator is used per instance. This is also the case in RAC environments.

> **See Also:** "How to View Scheduler Information" on page 28-8 for job coordinator administration and "Using the Scheduler in RAC Environments" on page 26-14 for RAC information

How Jobs Execute

When a job is picked for processing, the job slave:

1. Gathers all the metadata needed to run the job. As an example, arguments of the program and privilege information.
2. Starts a database session as the owner of the job, starts a transaction, and then starts executing the job.
3. Once the job is complete, the slave commits and ends the transaction.

4. Closes the session.

Job Slaves

Job slaves actually execute the jobs you submit. They are awakened by the job coordinator when it is time for a job to be executed. They gather metadata to run the job from the job table.

When a job is done, the slaves:

- update the entry in the job table to the `COMPLETED` status
- insert an entry into the job log table
- update the run count
- clean up
- look for new work (if none, they go to sleep)

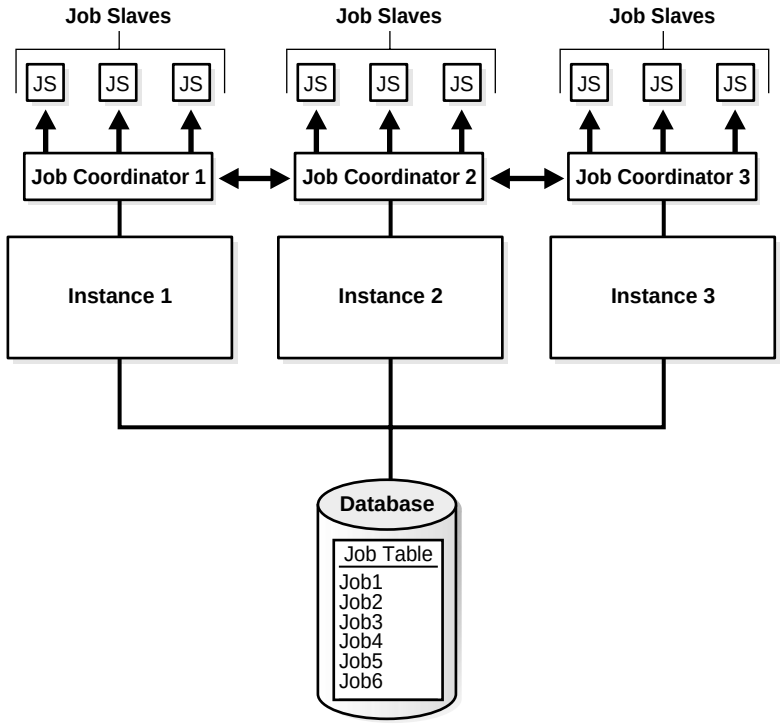
The Scheduler dynamically sizes the slave pool as required.

Using the Scheduler in RAC Environments

All Scheduler functionality performs the same when using RAC as in a single-instance environment. Just as in other environments, the Scheduler in a RAC environment uses one job table for each database and one job coordinator for each instance.

The job coordinators communicate with each other to keep information current. Figure 26–5 illustrates a typical RAC architecture, with each instance's job coordinator exchanging information with the others. Note that there is only one job table for the database.

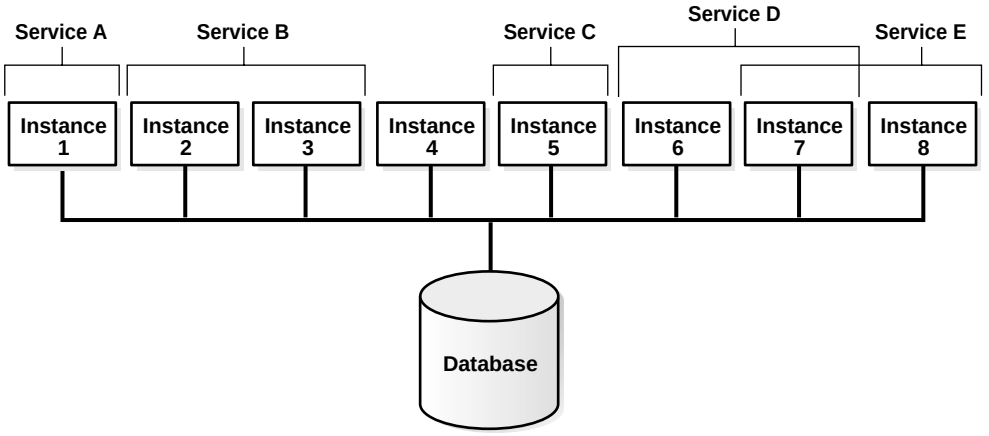
Figure 26–5 RAC Architecture and the Scheduler



Service Affinity when Using the Scheduler

The Scheduler enables you to specify the service on which a job should be run (service affinity). This ensures better availability than instance affinity because it guarantees that other nodes can be dynamically assigned to the service if an instance goes down. Instance affinity does not have this capability, so, when an instance goes down, none of the jobs with an affinity to that instance will be able to run until the instance comes back up. Figure 26–6 illustrates a typical example of how services and instances could be used.

Figure 26–6 Service Affinity and the Scheduler



In Figure 26–6, you could change the properties of the services and the Scheduler will automatically recognize the change.

Each job class maps to a service. If a service is not specified, the job class will belong to a default service.

27

Using the Scheduler

Oracle Database provides database scheduling capabilities through the Scheduler. This chapter introduces you to its use, and discusses the following topics:

- Scheduler Objects and Their Naming
- Administering Jobs
- Administering Programs
- Administering Schedules
- Administering Job Classes
- Administering Windows
- Administering Window Groups
- Allocating Resources Among Jobs

Scheduler Objects and Their Naming

Each Scheduler object is a complete database schema object of the form `[schema.]name`. Scheduler objects exactly follow the naming rules for database objects, so they must be unique in the SQL namespace.

When names for Scheduler objects are used in the `DBMS_SCHEDULER` package, SQL naming rules continue to be followed. By default, Scheduler object names are uppercase unless they are surrounded by double quotes. For example, when creating a job, its name must be provided. `job_name => 'my_job'` is the same as `job_name => 'My_Job'` and `job_name => 'MY_JOB'`, but not the same as `job_name => '"my_job"'`. These naming rules are also followed in those cases where comma-delimited lists of Scheduler object names are used within the `DBMS_SCHEDULER` package.

See *Oracle Database SQL Reference* for details regarding naming objects.

Administering Jobs

A job is the combination of a schedule and a program, along with any additional arguments required by the program. This section introduces you to basic job tasks, and discusses the following topics:

- Job Tasks and Their Procedures
- Creating Jobs
- Copying Jobs
- Altering Jobs
- Running Jobs
- Stopping Jobs
- Dropping Jobs
- Disabling Jobs
- Enabling Jobs

Job Tasks and Their Procedures

Table 27–1 illustrates common job tasks and their appropriate procedures and privileges:

Table 27–1 Job Tasks and Their Procedures

Task	Procedure	Privilege Needed
Create a job	`CREATE_JOB`	`CREATE JOB` or `CREATE ANY JOB`
Alter a job	`SET_ATTRIBUTE`	`ALTER` or `CREATE ANY JOB` or be the owner
Run a job	`RUN_JOB`	`ALTER` or `CREATE ANY JOB` or be the owner
Copy a job	`COPY_JOB`	`ALTER` or `CREATE ANY JOB` or be the owner
Drop a job	`DROP_JOB`	`ALTER` or `CREATE ANY JOB` or be the owner
Stop a job	`STOP_JOB`	`ALTER` or `CREATE ANY JOB` or be the owner
Disable a job	`DISABLE`	`ALTER` or `CREATE ANY JOB` or be the owner
Enable a job	`ENABLE`	`ALTER` or `CREATE ANY JOB` or be the owner

See "How to Manage Scheduler Privileges" on page 28-16 for further information regarding privileges.

Creating Jobs

You create jobs using the CREATE_JOB procedure. When creating a job, you must specify the action of the job, the schedule for the job, as well as some other attributes of the job. For example, the following statement creates a job called my_emp_job1, which is an insert into the sales table:

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB (
   job_name           =>  'my_emp_job1',
   job_type           =>  'PLSQL_BLOCK',
   job_action         =>  'INSERT INTO sales VALUES( 7987, ''SALLY'',
      ''ANALYST'', NULL, NULL, NULL, NULL, NULL);',
   start_date         =>  '28-APR-03 07.00.00 PM Australia/Sydney',
   repeat_interval    =>  'FREQ=DAILY;INTERVAL=2', /* every other day */
   end_date           =>  '20-NOV-04 07.00.00 PM Australia/Sydney',
   comments           =>  'My new job');
END;
/
```

You can create a job in another schema by specifying schema.job_name. The creator of a job is, therefore, not necessarily the job owner. The job owner is the user in whose schema the job is created, while the job creator is the user who is creating the job. Jobs are executed with the privileges of the schema in which the job is created. The NLS environment of the job when it runs is that which was present at the time the job was created.

Once a job is created, it can be queried using the *_SCHEDULER_JOBS views. Jobs are created disabled by default and they need to be enabled in order to be executed.

Job Attributes

Some job attributes are set at job creation time, while other job attributes are not. Instead, you can specify these attributes after the job has been created by using the SET_ATTRIBUTE procedure. See *PL/SQL Packages and Types Reference* for information about the SET_ATTRIBUTE procedure.

Setting Job Arguments

After creating a job, you may need to set job arguments. To set job arguments, use the SET_JOB_ARGUMENT_VALUE or SET_JOB_ANYDATA_VALUE procedures. Both

procedures have the same purpose, but SET_JOB_ANYDATA_VALUE is used for types that cannot be implicitly converted to and from VARCHAR2. A typical situation where you might want to set a job argument is for adding a new employee to a department. In this case, you might have a job that adds employees and assigns them the next available number in the department for a department ID. The following statement does this:

```
BEGIN
DBMS_SCHEDULER.SET_JOB_ARGUMENT_VALUE (
   job_name                => 'my_emp_job1',
   argument_position       => 2,
   argument_value          => 'John_Newman');
END;
/
```

If you use this procedure on an argument whose value has already been set, it will be overwritten. You can set argument values either by using the argument name or by the argument position. If a program is inlined, only setting by position is supported. Arguments are not supported for jobs of type `plsql_block`.

To remove a value that has been set, us the RESET_JOB_ARGUMENT procedure. This procedure can be used for both regular and anydata arguments.

See *PL/SQL Packages and Types Reference* for information about the procedures used in setting job arguments and their syntax.

Ways of Creating Jobs

You create a job using the CREATE_JOB procedure. Because this procedure is overloaded, there are several different ways of using it. In addition to inlining a job during the job creation, you can also create a job that points to a saved program and schedule. This is discussed in the following sections:

- Creating Jobs Using a Saved Program
- Creating Jobs Using a Saved Schedule
- Creating Jobs Using a Saved Program and Schedule

Creating Jobs Using a Saved Program You can also create a job by pointing to a saved program instead of inlining its action. To create a job using a saved program, you specify the value for `program_name` in the CREATE_JOB procedure when creating the job and do not specify the values for `job_type`, `job_action`, and `number_of_arguments`.

To use an existing program when creating a job, the owner of the job must be the owner of the program or have EXECUTE privileges on it. An example of using the CREATE_JOB procedure with a saved program is the following statement, which creates a job called my_new_job1:

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB (
   job_name          =>  'my_new_job1',
   program_name      =>  'my_saved_program',
   repeat_interval   =>  'FREQ=DAILY;BYHOUR=12',
   comments          =>  'Daily at noon');
END;
/
```

Creating Jobs Using a Saved Schedule You can also create a job by pointing to a saved schedule instead of inlining its schedule. To create a job using a saved schedule, you specify the value for schedule_name in the CREATE_JOB procedure when creating the job and do not specify the values for start_date, repeat_interval, and end_date.

You can use any saved schedule to create a job because all schedules are created with access to PUBLIC. An example of using the CREATE_JOB procedure with a saved schedule is the following statement, which creates a job called my_new_job2:

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB (
   job_name                =>  'my_new_job2',
   job_type                =>  'PLSQL_BLOCK',
   job_action              =>  'BEGIN foo(); END;',
   schedule_name           =>  'my_saved_schedule');
END;
/
```

Creating Jobs Using a Saved Program and Schedule A job can also be created by pointing to both a saved program and schedule. An example of using the CREATE_JOB procedure with a saved program and schedule is the following statement, which creates a new job called my_new_job3 based on the existing program my_saved_program1 and the existing schedule my_saved_schedule1:

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB (
   job_name           =>  'my_new_job3',
   program_name       =>  'my_saved_program1',
   schedule_name      =>  'my_saved_schedule1');
```

```
END;
/
```

Copying Jobs

You copy a job using the COPY_JOB procedure. This call copies all the attributes of the old job to the new job except the new job is created disabled and has another name.

Privileges Required for Copying a Job

You can copy a job if you are the owner of the job, or have ALTER privileges on the job, or have the CREATE ANY JOB privilege. Only SYS can copy a job from or into the SYS schema.

Altering Jobs

You alter a job using the SET_ATTRIBUTE procedure. All jobs can be altered, and, with the exception of the job name, all job attributes can be changed. If an enabled job is altered, the Scheduler will disable it, make the change and then reenable it. If any errors are encountered during the enable process, the job is not enabled and an error is generated. If there is a running instance of the job when the SET_ATTRIBUTE call is made, it is not affected by the call. The change is only seen in future runs of the job.

If any of the schedule attributes of a job are altered while the job is running, the next job run will be scheduled using the new schedule attributes. Schedule attributes of a job include schedule_name, start_date, end_date, and repeat_interval.

If any of the program attributes of a job are altered while the job is running, the new program attributes will take effect the next time the job runs. Program attributes of a job include program_name, job_action, job_type, and number_of_arguments. This is also the case for job argument values that have been set.

Granting ALTER on a job will let a user alter all attributes of that job except its program attributes (program_name, job_type, job_action, program_action, and number_of_arguments) and will not allow a user to use a PL/SQL expression to specify the schedule for a job.

In general, you should not alter a job that was automatically created for you by the database. Jobs that were created by the database have the column SYSTEM set to TRUE in several views. The attributes of a job are available in the *_SCHEDULER_JOB views.

It is perfectly valid for running jobs to alter their own job attributes using the SET_ATTRIBUTE procedure, however, these changes will not be picked up until the next scheduled run of the job.

See *PL/SQL Packages and Types Reference* for detailed information about the SET_ATTRIBUTE procedure and "Configuring the Scheduler" on page 28-1.

Running Jobs

Normally, jobs are executed asynchronously. The user creates a job and the API immediately returns and indicates whether the creation was successful. To find out whether the job succeeded, the user has to query the job table or the job log. While this is the expected behavior, for special cases, the database also allows users to execute jobs synchronously.

Running Jobs Asynchronously

You can schedule a job to run asynchronously based on the schedule defined when the job is created. In this case, the job is submitted to the job coordinator and is picked up by the job slaves for execution.

Running Jobs Synchronously

Once a job has been created, you can run the job synchronously using the RUN_JOB procedure with the use_current_session argument set to TRUE. In this case, the job will run within the user session that invoked the RUN_JOB call instead of being picked up by the coordinator and being executed by a job slave. To run the job using the RUN_JOB procedure, it must be enabled.

You can use the RUN_JOB procedure to test a job, thereby ensuring that it runs without errors. It can also be used to run a job outside of its specified schedule. For example, if an instance of a job failed because of some error. Once you fix the errors, you can use this procedure to run the job instead of scheduling a separate job for it.

Running a job using the RUN_JOB procedure with its use_current_session argument set to TRUE does not change the count for failure_count and run_count for the job. The job run will, however, be reflected in the job log. Runtime errors generated by the job are passed back to the invoker of RUN_JOB.

Job Run Environment

Jobs are run with the privileges that are granted to the job owner directly or indirectly through default logon roles. External OS roles are not supported. Given sufficient privileges, users can create jobs in other users' schemas. The creator and

the owner of the job can, therefore, be different. For example, if user jim has the CREATE ANY JOB privilege and creates a job in the scott schema, then the job will run with the privileges of scott.

The NLS environment of the session in which the job was created are saved and is used when the job is being executed. To alter the NLS environment in which a job runs, a job must be created in a session with different NLS settings.

Running External Jobs

An external job is a job that runs outside the database. All external jobs run as a low-privileged guest user, as has been determined by the database administrator while configuring external job support. Because the executable will be run as a low-privileged guest account, you should verify that it has access to necessary files and resources. Most, but not all, platforms support external jobs. For platforms that do not support external jobs, creating or setting the attribute of a job or a program to type EXECUTABLE returns an error. See your operating system-specific documentation for more information.

For an external job, job_type is specified as EXECUTABLE (If using named programs, the corresponding program_type would be EXECUTABLE). job_action (or corresponding program_action if using named programs) is the full OS-dependent path of the desired external executable plus optionally any command line arguments. For example, /usr/local/bin/perl or C:\perl\bin\perl. The program or job arguments for type EXECUTABLE must be a string type such as CHAR, VARCHAR2, or VARCHAR.

Some additional post-installation steps might be required to ensure that external jobs run as a low-privileged guest user. See your operating system-specific documentation for any post-installation configuration steps.

Setting Environment Variables for External Jobs To ensure that environment variables can be used with external jobs, you can use a wrapper such as Perl or sh before invoking the external job. As an example, if you have an external job (hello.exe) with USER_NAME and LOCATION environment variables, you could create a perl wrapper (hello.pl) such as the following:

```
$ENV { "USER_NAME" } = $ARGV[1];
$ENV { "LOCATION" } = $ARGV[2];
system ($ARGV[0]);
```

With a program_action of C:\home\hello.exe, the values would be C:\perl\bin\perl.exe (or /usr/local/bin/perl on UNIX, or

/home/mydir/bin/hello.pl, if hello.pl is executable). Your program arguments would be the following:

```
"C:\home\hello.exe"     --- the path where hello.exe is located
"Myname"                --- the value for USER_NAME
"Mytown"                --- the value for LOCATION
```

Stopping Jobs

You stop a running job using the STOP_JOB procedure. Job classes reside in the SYS schema, therefore, whenever job classes are used in comma-delimited lists, they must be preceded by SYS. For example, the following statement stops job1:

```
BEGIN
DBMS_SCHEDULER.STOP_JOB('job1');
END;
/
```

Any instance of the job will be stopped. After stopping the job, the state of a one-time job will be set to STOPPED whereas the state of a repeating job will be set to SCHEDULED because the next run of the job is scheduled.

The Scheduler tries to gracefully stop the job using an interrupt mechanism. This method gives control back to the slave process, which can update the status of the job to STOPPED.

If the interrupt is not successful, the STOP_JOB call will fail. The job will be stopped as soon as possible after its current uninterruptable operation is done. Users with the MANAGE SCHEDULER privilege can force the job to stop sooner by setting the force option to TRUE in the STOP_JOB call. In this case, the call forcibly terminates the slave process that was running the job, thus stopping the job.

The STOP_JOB procedure accepts job_name as an argument. This can be the name of a job or a comma-delimited list of job names. It can also be the name of a job class or a list of job class names. For example, the following statement combines both jobs and job classes:

```
BEGIN
DBMS_SCHEDULER.STOP_JOB ('job1, job2, job3,
   sys.jobclass1, sys.jobclass2, sys.jobclass3');
END;
/
```

If the name of a job class is specified using STOP_JOB, the jobs that belong to that job class are stopped. The job class is not affected by this call.

Only running jobs can be stopped. Stopping a job that is not running generates a PL/SQL exception saying that the job is not running. Stopping a job that does not exist also causes an error. When a list of job names is provided, the Scheduler stops executing the list of jobs on the very first job that returns an error.

Caution: When a job is stopped, only the current transaction will be rolled back. Note that if there are commits in the executable that the job is running, then only the current transaction will be rolled back. This can cause data inconsistency.

Dropping Jobs

You drop a job using the DROP_JOB procedure. Dropping a job results in the job being removed from the job table, its metadata being removed, and it no longer being visible in the *_SCHEDULER_JOBS views. Therefore, no more runs of the job will be executed.

If an instance of the job is running at the time of the call, the call results in an error. You can still drop the job by setting the `force` option in the call to TRUE. Setting the `force` option to TRUE attempts to first stop (issues the STOP_JOB call) the running job instance and then drop the job. By default, `force` is set to FALSE. If the user does not have privileges to stop the job, the DROP_JOB call will fail.

The DROP_JOB procedure accepts `job_name` as an argument. This can be the name of a job or a comma-delimited list of job names. It can also be the name of a job class or a list of job class names. For example, the following statement combines both jobs and job classes:

```
BEGIN
DBMS_SCHEDULER.DROP_JOB ('job1, job2, job3,
   sys.jobclass1, sys.jobclass2, sys.jobclass3');
END;
/
```

If the name of a job class is specified in this procedure call, the jobs that belong to that job class are dropped, but the job class itself is not dropped. The DROP_JOB_CLASS procedure should be used to drop the job class. See "Dropping Job Classes" on page 27-25 for information about how to drop job classes.

Attempting to drop a job or job class that does not exist generates an error stating that the object does not exist. If a list of job names is specified in the DROP_JOB call, the call fails on the first job that cannot be dropped. In the preceding example, if `job2` could not be dropped, the DROP_JOB call fails. `job1` will be dropped but it

will not be attempted to drop the other jobs in the list. The error returned by the Scheduler will contain the name of the job that caused the error.

Disabling Jobs

You disable a job using the DISABLE procedure. A job can also become disabled for other reasons. For example, a job will be disabled when the job class it belongs to is dropped. A job is also disabled if either the program or the schedule that it points to is dropped. Note that if the program or schedule that the job points to is disabled, the job will not be disabled and will therefore result in an error when the Scheduler tries to execute the job.

Disabling a job means that, although the metadata of the job is there, it should not run and the job coordinator will not pick up these jobs for processing. When a job is disabled, its `state` in the job table is changed to `disabled`.

When a job is disabled with the `force` option set to FALSE and the job is currently running, an error is returned. When `force` is set to TRUE, the job is disabled, but the currently running instance is allowed to finish.

You can also disable several jobs in one call by providing a comma-delimited list of job names or job class names to the DISABLE procedure call. For example, the following statement combines jobs with job classes:

```
BEGIN
DBMS_SCHEDULER.DISABLE('job1, job2, job3, sys.jobclass1, sys.jobclass2');
END;
/
```

Note that if a list of job class names is provided, the jobs in the job class are disabled.

Note that if it is not possible to disable `job2`, then the DISABLE call will fail. `job1` will be disabled but `job2`, `job3`, and jobs in `jobclass1`, and `jobclass2` will not be disabled.

Enabling Jobs

You enable jobs by using the ENABLE procedure. The effect of using this procedure is that the job will now be picked up by the job coordinator for processing. Jobs are created disabled by default, so you need to enable them before they can run. When a job is enabled, a validity check is performed. If the check fails, the job is not enabled.

You can enable several jobs in one call by providing a comma-delimited list of job names or job class names to the ENABLE procedure call. For example, the following statement combines jobs with job classes:

```
BEGIN
DBMS_SCHEDULER.ENABLE ('job1, job2, job3,
   sys.jobclass1, sys.jobclass2, sys.jobclass3');
END;
/
```

Note that if a list of job class names is provided, the jobs in the job class are enabled. Also, if it is not possible to enable `job2`, then the ENABLE call will fail. `job1` will be enabled but `job2`, `job3`, `jobclass1`, and `jobclass2` will not be enabled.

Administering Programs

A program is a collection of metadata about a particular task. This section introduces you to basic program tasks, and discusses the following topics:

- Program Tasks and Their Procedures
- Creating Programs
- Altering Programs
- Dropping Programs
- Disabling Programs
- Enabling Programs

Program Tasks and Their Procedures

Table 27–2 illustrates common program tasks and their appropriate procedures and privileges:

Table 27–2 Program Tasks and Their Procedures

Task	Procedure	Privilege Needed
Create a program	CREATE_PROGRAM	CREATE JOB or CREATE ANY JOB
Alter a program	SET_ATTRIBUTE	ALTER or CREATE ANY JOB or be the owner
Drop a program	DROP_PROGRAM	ALTER or CREATE ANY JOB or be the owner
Disable a program	DISABLE	ALTER or CREATE ANY JOB or be the owner

Table 27–2 (Cont.) Program Tasks and Their Procedures

Task	Procedure	Privilege Needed
Enable a program	ENABLE	ALTER or CREATE ANY JOB or be the owner

See "How to Manage Scheduler Privileges" on page 28-16 for further information regarding privileges.

Creating Programs

You create programs by using the CREATE_PROGRAM procedure. By default, programs are created in the schema of the creator. To create a program in another user's schema, you need to qualify the program name with the schema name. For other users to use your programs, they must have EXECUTE privileges on the program, therefore, once a program has been created, you have to grant the EXECUTE privilege on it. An example of creating a program is the following, which creates a program called my_program1:

```
BEGIN
DBMS_SCHEDULER.CREATE_PROGRAM (
   program_name           => 'my_program1',
   program_action         => '/usr/local/bin/date',
   program_type           => 'EXECUTABLE',
   comments               => 'My comments here');
END;
/
```

> **See Also:** *PL/SQL Packages and Types Reference* for detailed information about the SET_ATTRIBUTE procedure and "Configuring the Scheduler" on page 28-1

Defining Program Arguments

After creating a program, you will want to define program arguments when planning its execution. All arguments must be defined before the program can be enabled.

To set program argument values, use the DEFINE_PROGRAM_ARGUMENT or DEFINE_ANYDATA_ARGUMENT procedures. Both procedures have the same purpose, but DEFINE_ANYDATA_ARGUMENT is used for types that cannot be converted to VARCHAR2. A typical situation where you might want to define a program argument is for adding a new employee to a company. In this case, you

might have a job that adds employees and assigns them the next available number in the company for an employee ID. The following statement does this:

```
BEGIN
DBMS_SCHEDULER.DEFINE_PROGRAM_ARGUMENT (
   program_name            => 'my_program2',
   argument_position       => 2,
   argument_name           => 'ename',
   argument_type           => 'VARCHAR2',
   default_value           => 'N/A');
END;
/
```

You can drop a program argument either by name or by position, as in the following statements:

```
BEGIN
DBMS_SCHEDULER.DROP_PROGRAM_ARGUMENT (
   program_name            => 'my_program2',
   argument_position       => 2);

DBMS_SCHEDULER.DROP_PROGRAM_ARGUMENT (
   program_name            => 'my_program2',
   argument_name           => 'ename');
END;
/
```

In some special cases, program logic is dependent on the Scheduler environment. The Scheduler has some predefined metadata arguments that can be passed as an argument to the program for this purpose. For example, for some jobs whose schedule is a window name, it is useful to know how much longer the window will be open when the job is started. This is possible by defining the window end time as a metadata argument to the program.

If a program needs access to specific job metadata, you can define a special metadata argument using the `DEFINE_METADATA_ARGUMENT` procedure, so values will be filled in by the Scheduler when the program is executed. You can set the following arguments:

- `job_name`
- `job_owner`
- `job_start`
- `window_start`

- window_end

See Also: *PL/SQL Packages and Types Reference* for detailed information about the SET_ATTRIBUTE and DEFINE_METADATA_ARGUMENT procedures and "Configuring the Scheduler" on page 28-1

Altering Programs

You alter programs by using the SET_ATTRIBUTE or SET_ATTRIBUTE_NULL procedure. With the exception of program name, all program attributes can be changed. If any currently running jobs use the program that is altered, they will continue to run with the program definition prior to the alter. The job will run with the new program definition the next time the job executes.

When a program is altered and it was in the enabled state, the Scheduler first disables it, applies the change, and then reenables it. If any errors are encountered during the enable process, the program is not reenabled and an error is generated.

The SET_ATTRIBUTE_NULL is only required for setting the value of any program attribute to NULL.

Dropping Programs

You drop a program using the DROP_PROGRAM procedure. You can also drop several programs in one call by providing a comma-delimited list of program names to the procedure. For example, the following statement drops all three programs:

```
BEGIN
DBMS_SCHEDULER.DROP_PROGRAM('program1, program2, program3');
END;
/
```

Dropping a program that does not exist generates an error stating that the program does not exist. Note that if it is not possible to drop program2 in the preceding example, then the DROP_PROGRAM call fails. program1 will be dropped but program2 and program3 are not dropped.

If there are jobs that point to the program that you are trying to drop, you will not be able to drop the program unless you set force to TRUE in the procedure call. By default, force is set to FALSE. When the program is dropped, those jobs that point to the program will be disabled.

Running jobs that point to the program are not affected by the DROP_PROGRAM call, and are allowed to continue.

Any arguments that pertain to the program are also dropped when the program is dropped.

Disabling Programs

You disable a program using the DISABLE procedure. When a program is disabled, the status is changed to disabled. A disabled program implies that, although the metadata is still there, jobs that point to this program cannot run.

You can also disable several programs in one call by providing a comma-delimited list of program names to the DISABLE procedure call. For example, the following statement disables all three programs:

```
DBMS_SCHEDULER.DISABLE('program1, program2, program3');
```

Disabling a program that does not exist generates an error stating that the program does not exist. Note that if it is not possible to disable program2 in this example, then the DISABLE call will fail. program1 will be disabled but program2 and program3 will not be disabled.

If there are jobs that point to the program that you are trying to disable, you will not be able to disable the program unless you set force to TRUE in the procedure call. By default, force is set to FALSE. When the program is disabled, those jobs that point to the program, will not be disabled, however, the job will fail at runtime because its program will not be valid.

Running jobs that point to the program are not affected by the DISABLE call, and are allowed to continue.

Any argument that pertains to the program will not be affected when the program is disabled.

A program can also become disabled for other reasons. For example, if a program argument is dropped or number_of_arguments is changed so that all arguments are no longer defined.

Enabling Programs

You enable a program using the ENABLE procedure. When a program is enabled, the enabled flag is set to TRUE. Programs are created disabled by default, therefore, you have to enable them before you can enable jobs that point to them. Before

programs are enabled, validity checks are performed to ensure that the action is valid and that all arguments are defined.

You can enable several programs in one call by providing a comma-delimited list of program names to the ENABLE procedure call. For example, the following statement enables three programs:

```
BEGIN
DBMS_SCHEDULER.ENABLE('program1, program2, program3');
END;
/
```

Enabling a program that does not exist will cause an error stating that the program does not exist. Note that if it is not possible to enable `program2` in this example, then the ENABLE call will fail. `program1` will be enabled but `program2` and, `program3` will not be enabled.

Administering Schedules

A schedule defines when a job should be run or when a window should open. Schedules can be saved. Schedules can be shared among users by creating and saving them as an object in the database.

This section introduces you to basic schedule tasks, and discusses the following topics:

- Schedule Tasks and Their Procedures
- Creating Schedules
- Altering Schedules
- Dropping Schedules
- Setting the Repeat Interval

Schedule Tasks and Their Procedures

Table 27–3 illustrates common schedule tasks and the procedures you use to handle them.

Table 27–3 Schedule Tasks and Their Procedures

Task	Procedure	Privilege Needed
Create a schedule	CREATE_SCHEDULE	CREATE JOB or CREATE ANY JOB

Table 27–3 (Cont.) Schedule Tasks and Their Procedures

Task	Procedure	Privilege Needed
Alter a schedule	SET_ATTRIBUTE	ALTER or CREATE ANY JOB or be the owner
Drop a schedule	DROP_SCHEDULE	ALTER or CREATE ANY JOB or be the owner

See "How to Manage Scheduler Privileges" on page 28-16 for further information regarding privileges.

Creating Schedules

You create schedules by using the CREATE_SCHEDULE procedure. Schedules are created in the schema of the user creating the schedule, and are enabled when first created. You can create a schedule in another user's schema. Once a schedule has been created, it can be used by other users. The schedule is created with access to PUBLIC. Therefore, there is no need to explicitly grant access to the schedule. An example of creating a schedule is the following statement:

```
BEGIN
DBMS_SCHEDULER.CREATE_SCHEDULE (
  schedule_name     => 'my_stats_schedule',
  start_date        => SYSTIMESTAMP,
  end_date          => SYSTIMESTAMP + INTERVAL '30' day,
  repeat_interval   => 'FREQ=HOURLY; INTERVAL=4',
  comments          => 'Every 4 hours');
END;
/
```

Altering Schedules

You alter a schedule by using the SET_ATTRIBUTE procedure. Altering a schedule changes the definition of the schedule. With the exception of schedule name, all attributes can be changed. The attributes of a job are available in the *_SCHEDULER_SCHEDULES views.

If a schedule is altered, the change will not affect running jobs and open windows that use this schedule. The change will only be in effect the next time the jobs runs or the window opens.

Dropping Schedules

You drop a schedule using the DROP_SCHEDULE procedure. This procedure call will delete the schedule object from the database. You can also drop several schedules in one call by providing a comma-delimited list of schedule names to the DROP_SCHEDULE procedure call. For example, the following statement drops three schedules:

```
BEGIN
DBMS_SCHEDULER.DROP_SCHEDULE('schedule1, schedule2, schedule3');
END;
/
```

If there are jobs or windows that use this schedule, the DROP_SCHEDULE call will fail unless you set `force` to TRUE. If you set `force` to TRUE, then the jobs and windows that use this schedule will be disabled before the schedule is dropped.

Setting the Repeat Interval

You control how often a job repeats by setting the `repeat_interval` attribute. The expression specified is evaluated to determine the next time the job should run. If no value for `repeat_interval` is specified, the job will run only once at the specified start date.

Immediately after a job is started, the `repeat_interval` is evaluated to determine the next scheduled execution time of the job. It is possible that the next scheduled execution time arrives while the job is still running. A new instance of the job, however, will not be started until the current one completes.

There are two ways to specify the repeat interval:

- Using the Scheduler Calendaring Syntax
- Using a PL/SQL Expression

Using the Scheduler Calendaring Syntax

The primary method of setting how often a job will repeat is by setting the `repeat_interval` attribute with Oracle Database calendaring expression. See *PL/SQL Packages and Types Reference* for a detailed description of the calendaring syntax for `repeat_interval` as well as the CREATE_SCHEDULE procedure.

Examples of Using Calendaring Expressions

The following examples illustrate simple tasks.

Execute every Friday.

```
FREQ=WEEKLY; BYDAY=FRI;
```

Execute every other Friday.

```
FREQ=WEEKLY; INTERVAL=2; BYDAY=FRI;
```

Execute on the last day of every month.

```
FREQ=MONTHLY; BYMONTHDAY-1;
```

Execute on the next to last day of every month.

```
FREQ=MONTHLY; BYMONTHDAY-2;
```

Execute on March 10th.

```
FREQ=YEARLY; BYMONTH=MAR; BYMONTHDAY=10;
```

Execute every 10 days.

```
FREQ=DAILY; INTERVAL=10;
```

Execute daily at 4, 5, and 6PM.

```
FREQ=DAILY; BYHOUR=16,17,18;
```

Execute on the 15th day of every other month.

```
FREQ=MONTHLY; INTERVAL=2; BYMONTHDAY=15;
```

Execute on the 29th day of every month.

```
FREQ=MONTHLY; BYMONTHDAY=29;
```

Execute on the second Wednesday of each month.

```
FREQ=MONTHLY; BYDAY=2WED;
```

Execute on the last Friday of the year.

```
FREQ=YEARLY; BYDAY=-1FRI;
```

Execute every 50 hours.

```
FREQ=HOURLY; INTERVAL=50;
```

Execute on the last day of every other month.

```
FREQ=MONTHLY; INTERVAL=2; BYMONTHDAY-1;
```

Execute hourly for the first three days of every month.

```
FREQ=HOURLY; BYMONTHDAY=1,2,3;
```

A repeat interval of "`FREQ=MINUTELY;INTERVAL=2;BYHOUR=17; BYMINUTE=2,4,5,50,51,7;`" with a start date of 28-FEB-2004 23:00:00 will generate the following schedule:

```
SUN 29-FEB-2004 17:02:00
SUN 29-FEB-2004 17:04:00
SUN 29-FEB-2004 17:50:00
MON 01-MAR-2004 17:02:00
MON 01-MAR-2004 17:04:00
MON 01-MAR-2004 17:50:00
...
```

A repeat interval of "`FREQ=MONTHLY;BYMONTHDAY=15, -1`" with a start date of 29-DEC-2003 9:00:00 will generate the following schedule:

```
WED 31-DEC-2003 09:00:00
THU 15-JAN-2004 09:00:00
SAT 31-JAN-2004 09:00:00
SUN 15-FEB-2004 09:00:00
SUN 29-FEB-2004 09:00:00
MON 15-MAR-2004 09:00:00
WED 31-MAR-2004 09:00:00
...
```

As an example of using the calendaring syntax, consider the following statement:

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB (
   job_name             => 'scott.my_job1',
   start_date           => '15-JUL-03 01.00.00 AM Europe/Warsaw',
   repeat_interval      => 'FREQ=MINUTELY; INTERVAL=30;',
   end_date             => '15-SEP-04 01.00.00 AM Europe/Warsaw',
   comments             => 'My comments here');
END;
/
```

This creates `my_job1` in `scott`. It will run for the first time on July 15th and then run until September 15. The job is run every 30 minutes.

Using a PL/SQL Expression

When you need more complicated capabilities than the calendaring syntax provides, you can use PL/SQL expressions. You cannot, however, use PL/SQL expressions for windows or named schedules. The PL/SQL expression must evaluate to a date or a timestamp. Other than this restriction, there are no limitations, so with sufficient programming, you can create every possible repeat interval. As an example, consider the following statement:

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB (
   job_name            => 'scott.my_job2',
   start_date          => '15-JUL-03 01.00.00 AM Europe/Warsaw',
   repeat_interval     =>  'SYSTIMESTAMP + INTERVAL '30' MINUTE',
   end_date            => '15-SEP-04 01.00.00 AM Europe/Warsaw',
   comments            => 'My comments here');
END;
/
```

This creates `my_job1` in `scott`. It will run for the first time on July 15th and then every 30 minutes until September 15. The job is run every 30 minutes because `repeat_interval` is set to `SYSTIMESTAMP + INTERVAL '30' MINUTE`, which returns a date 30 minutes into the future.

Differences Between PL/SQL Expression and Calendaring Syntax Behavior

The following are important differences in behavior between a calendering expression and PL/SQL repeat interval:

- Start date

  Using the calendaring syntax, the start date is a reference date only. This means that the schedule is valid as of this date. It does not mean that the job will start on the start date.

  Using a PL/SQL expression, the start date represents the actual time that the job will start executing for the first time.

- Next run time

  Using the calendaring syntax, the next time the job will execute is fixed.

  Using the PL/SQL expression, the next time the job will execute depends on the actual start time of the current run of the job. As an example of the difference, if a job started at 2:00 PM and its schedule was to repeat every 2 hours, then, if the repeat interval was specified with the calendaring syntax, it would repeat at 4, 6 and so on. If PL/SQL was used and the job started at 2:10, then the job would

repeat at 4:10, and if the next job actually started at 4:11, then the subsequent run would be at 6:11.

To illustrate these two points, consider a situation where you have a start date of 15-July-2003 1:45:00 and you want it to repeat every two hours. A calendar expression of "`FREQ=HOURLY; INTERVAL=2; BYMINUTE=0;`" will generate the following schedule:

```
TUE 15-JUL-2003  03:00:00
TUE 15-JUL-2003  05:00:00
TUE 15-JUL-2003  07:00:00
TUE 15-JUL-2003  09:00:00
TUE 15-JUL-2003  11:00:00
...
```

Note that the calendar expression repeats every two hours on the hour.

A PL/SQL expression of "`SYSTIMESTAMP + interval '2' hour`", however, might have a run time of the following:

```
TUE 15-JUL-2003  01:45:00
TUE 15-JUL-2003  03:45:05
TUE 15-JUL-2003  05:45:09
TUE 15-JUL-2003  07:45:14
TUE 15-JUL-2003  09:45:20
...
```

Repeat Intervals and Daylight Savings

For repeating jobs, the next time a job is scheduled to execute is stored in a timestamp with time zone column. When using the calendaring syntax, the time zone is retrieved from `start_date`. For more information on what happens when `start_date` is not specified, see *PL/SQL Packages and Types Reference*.

In the case of repeat intervals that are based on PL/SQL expressions, the time zone is part of the timestamp that is returned by the PL/SQL expression. In both cases, it is important to use region names. For example, `"Europe/Istanbul"`, instead of absolute time zone offsets such as `"+2:00"`. Only when a time zone is specified as a region name will the Scheduler follow daylight savings adjustments that apply to that region.

Administering Job Classes

Jobs can be difficult to manage on an individual basis, so the Scheduler uses job classes, which group jobs with common characteristics and behavior together. Prioritization among job classes is possible using resource plans.

There is a default job class that is created with the database. If you create a job without specifying a job class, the job will be assigned to this default job class (DEFAULT_JOB_CLASS). The default job class has the EXECUTE privilege granted to PUBLIC so any database user who has the privilege to create a job can create a job in the default job class. Job classes are created in the SYS schema.

This section introduces you to basic job class tasks, and discusses the following topics:

- Job Class Tasks and Their Procedures
- Creating Job Classes
- Altering Job Classes
- Dropping Job Classes

Job Class Tasks and Their Procedures

Table 27–4 illustrates common job class tasks and their appropriate procedures and privileges:

Table 27–4 Job Class Tasks and Their Procedures

Task	Procedure	Privilege Needed
Create a job class	CREATE_JOB_CLASS	MANAGE SCHEDULER
Alter a job class	SET_ATTRIBUTE	MANAGE SCHEDULER
Drop a job class	DROP_JOB_CLASS	MANAGE SCHEDULER

See "How to Manage Scheduler Privileges" on page 28-16 for further information regarding privileges.

Creating Job Classes

You create a job class using the CREATE_JOB_CLASS procedure. For example, the following statement creates a job class for all finance jobs:

```
BEGIN
```

```
DBMS_SCHEDULER.CREATE_JOB_CLASS (
   job_class_name             =>  'finance_jobs',
   resource_consumer_group    =>  'finance_group');
END;
/
```

To query job classes, use the *_SCHEDULER_JOB_CLASSES views.

Job classes are created in the SYS schema. For users to create jobs that belong to a job class, the job owner must have EXECUTE privileges on the job class. Therefore, after the job class has been created, EXECUTE privileges must be granted on the job class so that users create jobs belonging to that class. You can also grant the EXECUTE privilege to a role.

See *PL/SQL Packages and Types Reference* for detailed information about the SET_ATTRIBUTE procedure and "Configuring the Scheduler" on page 28-1 for examples of creating job classes.

Altering Job Classes

You can alter a job class by using the SET_ATTRIBUTE procedure. With the exception of the default job class, all job classes can be altered. Other than the job class name, all the attributes of a job class can be altered. The attributes of a job class are available in the *_SCHEDULER_JOB_CLASSES views.

When a job class is altered, running jobs that belong to the class are not affected. The change only takes effect for jobs that have not started running yet.

See *PL/SQL Packages and Types Reference* for detailed information about the SET_ATTRIBUTE procedure and "Configuring the Scheduler" on page 28-1.

Dropping Job Classes

You can drop a job class using the DROP_JOB_CLASS procedure. Dropping a job class means that all the metadata about the job class is removed from the database.

If there are jobs that belong to this job class, the DROP_JOB_CLASS call generates an error. The job class can still be dropped by setting the `force` option to TRUE, in which case the jobs belonging to the class are disabled and moved to the default class. If you drop a job class with a running job, the job continues running.

You can also drop several job classes in one call by providing a comma-delimited list of job class names to the DROP_JOB_CLASS procedure call. For example, the following statement drops three job classes:

```
BEGIN
DBMS_SCHEDULER.DROP_JOB_CLASS('jobclass1, jobclass2, jobclass3');
END;
/
```

Dropping a job class that does not exist will generate an error that will list the name of the job class that failed. Note that if it is not possible to drop `jobclass2`, then the DROP_JOB_CLASS call will fail. `jobclass1` will be dropped but `jobclass2` and `jobclass3` will not be dropped.

Administering Windows

Windows provide you with the functionality to activate different resource plans at different times. A resource plan is a component of the Resource Manager, which enables users to prioritize resources (most notably CPU) among resource consumer groups. The priorities are specified in a resource plan.

Each job class points to a resource consumer group and the same resource plan can thus be used to manager priorities among job classes. The next step is to provide different resource allocations at different times. For example, during the day, the finance group gets more resources, but at night, the admin group gets more resources.

The key attributes of windows are their:

- schedules

  These control when the window is in effect.

- durations

  These control how long the window is open.

- resource plans

  These control the resource priorities among the job classes

Only one window can be in effect at any given time, and a window is described as open if it is in effect. There is only one resource plan active for each window. Running jobs can see a change in the resources that are allocated to them when there is a change in resource plans. All window activity is written to the window log. Windows belong to the SYS schema.

This section introduces you to basic window tasks, and discusses the following topics:

- Window Tasks and Their Procedures

Window Tasks and Their Procedures

Table 27–5 illustrates common window tasks and the procedures you use to handle them.

Table 27–5 Window Tasks and Their Procedures

Task	Procedure	Privilege Needed
Create a window	CREATE_WINDOW	MANAGE SCHEDULER
Open a window	OPEN_WINDOW	MANAGE SCHEDULER
Close a window	CLOSE_WINDOW	MANAGE SCHEDULER
Alter a window	SET_ATTRIBUTE	MANAGE SCHEDULER
Drop a window	DROP_WINDOW	MANAGE SCHEDULER
Disable a window	DISABLE	MANAGE SCHEDULER
Enable a window	ENABLE	MANAGE SCHEDULER

See "How to Manage Scheduler Privileges" on page 28-16 for further information regarding privileges.

Creating Windows

You create windows by using the CREATE_WINDOW procedure. When creating a window, you can specify the schedule for the window. Alternatively, you can also create a window that points to a saved schedule instead of inlining it during the window creation. The following statement creates a window called my_window1

that uses a resource plan of my_resourceplan1 and repeats every midnight for an hour:

```
BEGIN
DBMS_SCHEDULER.CREATE_WINDOW (
   window_name       => 'my_window1',
   start_date        => '01-JAN-03 12:00:00AM',
   repeat_interval   => 'FREQ=DAILY',
   resource_plan     => 'my_resourceplan1',
   duration          => interval '60' minute,
   comments          => 'My window');
END;
/
```

Windows are created in the SYS schema. The Scheduler does not check if there is already a window defined for that schedule. Therefore, this may result in windows that overlap.

Creating Windows Using a Saved Schedule

You can also create a window by pointing to a saved schedule instead of inlining its schedule. To create a window using a saved schedule, use the version of the CREATE_WINDOW procedure that has the schedule_name argument.

You can use any saved schedule to create a window because all schedules are created with access to public. For example, the following statement creates a window with a saved schedule:

```
BEGIN
DBMS_SCHEDULER.CREATE_WINDOW (
   window_name         =>  'my_window100',
   schedule_name       =>  'my_stats_schedule',
   resource_plan       =>  'my_resourceplan1',
   duration            =>  interval '160' minute,
   comments            =>  'My window');
END;
/
```

Using a saved schedule that has a PL/SQL expression as its repeat interval is not supported for windows. The CREATE_WINDOW call will fail in this case.

See *PL/SQL Packages and Types Reference* for further details about CREATE_WINDOW and "Configuring the Scheduler" on page 28-1.

Altering Windows

You alter a window using the SET_ATTRIBUTE procedure. With the exception of WINDOW_NAME, all the attributes of a window can be changed when it is altered. The attributes of a window are available in the *_SCHEDULER_WINDOWS views.

When a window is altered, it does not affect an active window. The changes only take effect the next time the window opens.

All windows can be altered. If you alter a window that is disabled, it will remain disabled after it is altered. An enabled window will be automatically disabled, altered, and then reenabled, if the validity checks performed during the enable process are successful.

See *PL/SQL Packages and Types Reference* for detailed information about the SET_ATTRIBUTE procedure and "Configuring the Scheduler" on page 28-1.

Opening Windows

When a window opens, the Scheduler switches to the resource plan that has been associated with it during its creation. If there are jobs running when the window opens, the resources allocated to them might change due to the switch in resource plan.

There are two ways a window can open:

- Based on a schedule

  A window will open based on the schedule that is defined during its creation.

- Manually using the OPEN_WINDOW procedure

  This procedure opens the window independent of its schedule. This window will open and the resource plan associated with it will take effect immediately. Only an enabled window can be manually opened.

  In the OPEN_WINDOW procedure, you can specify the time interval that the window should be open for, using the duration attribute. The duration is of type interval day to second. If the duration is not specified, then the window will be opened for the regular duration as stored with the window.

  Opening a window manually has no impact on regular scheduled runs of the window.

  When a window that was manually opened closes, the rules about overlapping windows are applied to determine which other window should be opened at that time if any at all.

If you try to open a window that does not exist, an error is generated. If you try to open a window, and there is already an open window, then you will get an error. However, you can force a window to open even if there is one already open by setting the `force` option to `TRUE` in the `OPEN_WINDOW` call.

When the `force` option is set to `TRUE`, the Scheduler automatically closes any window that is open at that time, even if it has a higher priority. For the duration of this manually opened window, the Scheduler does not open any other scheduled windows even if they have a higher priority. You can open a window that is already open. In this case, the window stays open for the duration specified in the call, from the time the `OPEN_WINDOW` command was issued.

Consider an example to illustrate this. `window1` was created with a duration of four hours. It has how been open for two hours. If at this point you reopen `window1` using the `OPEN_WINDOW` call and do not specify a duration, then `window1` will be open for another four hours because it was created with that duration. If you specified a duration of 30 minutes, the window will close in 30 minutes.

A window can fail to open if the resource plan has been manually switched using the `ALTER SYSTEM` statement with the `force` option.

When a window is opened, there will be an entry in the `*_SCHEDULER_WINDOW_LOG` views to indicate this.

Closing Windows

There are two ways a window can close:

- Based on a schedule

  A window will close based on the schedule defined at creation time.

- Manually, using the `CLOSE_WINDOW` procedure

  The `CLOSE_WINDOW` procedure will close an open window prematurely.

A closed window means that it is no longer in effect. When a window is closed, the Scheduler will switch the resource plan to the one that was in effect outside the window or in the case of overlapping windows to another window. If you try to close a window that does not exist or is not open, an error is generated.

A job that is running will not close when the window it is running in closes unless the attribute `stop_on_window_close` was set to `TRUE` when the job was created.

However, the resources allocated to the job may change because the resource plan may change.

When a running job has a window group as its schedule, the job will not be stopped when its window is closed if another window that is also a member of the same window group then becomes active. This is the case even if the job was created with the attribute `stop_on_window_close` set to `TRUE`.

When a window is closed, an entry will be added to the window log `DBA_SCHEDULER_WINDOW_LOG`.

Dropping Windows

You drop a window using the `DROP_WINDOW` procedure. When a window is dropped, all metadata about the window is removed from the `*_SCHEDULER_WINDOWS` views. All references to the window are removed from window groups.

You can also drop several windows in one call by providing a comma-delimited list of window names or window group names to the `DROP_WINDOW` procedure. For example, the following statement drops both windows and window groups:

```
BEGIN
DBMS_SCHEDULER.DROP_WINDOW ('window1, window2,
  window3, windowgroup1, windowgroup2');
END;
/
```

Note that if a window group name is provided, then the windows in the window group are dropped, but the window group is not dropped. To drop the window group, you must use the `DROP_WINDOW_GROUP` procedure.

Note that if it is not possible to drop `window2` in this example, then the `DROP_WINDOW` call will fail. `window1` will be dropped but `window2`, `window3`, `windowgroup1`, and `windowgroup2` will not be dropped.

If the window is open, the `DROP_WINDOW` call generates an error unless the `force` option is set to `TRUE` in the procedure call. If this is the case, the window will be closed then dropped. When the window is closed, normal close window rules apply.

If there are jobs that have the window as their schedule, you will not be able to drop the window unless you set `force` to `TRUE` in the procedure call. By default, `force` is set to `FALSE`. When the window is dropped, those jobs that have the window as their schedule will be disabled. However, jobs that have a window group of which the dropped window was a member as their schedule will not be disabled.

Running jobs that have the window as their schedule will be allowed to continue, unless the stop_on_window_close flag was set to TRUE when the job was created. If this is the case, the job will be stopped when the window is dropped.

Disabling Windows

You disable a window using the DISABLE procedure. This means that the window will not open, however, the metadata of the window is still there, so it can be reenabled. Because the DISABLE procedure is used for several Scheduler objects, when disabling windows, they must be preceded by SYS.

Disabling a window that is open will cause an error unless the force option is set to TRUE in the procedure call. If force is set to TRUE, disabling a window that is open will succeed but will not close the window. It will prevent the window from opening in the future until it is reenabled.

A window can also become disabled for other reasons. For example, a window will become disabled when it is at the end of its schedule. Also, if a window points to a schedule that no longer exists, it becomes disabled.

If there are jobs that have the window as their schedule, you will not be able to disable the window unless you set force to TRUE in the procedure call. By default, force is set to FALSE. When the window is disabled, those jobs that have the window as their schedule will not be disabled.

You can disable several windows in one call by providing a comma-delimited list of window names or window group names to the DISABLE procedure call. For example, the following statement disables both windows and window groups:

```
BEGIN
DBMS_SCHEDULER.DISABLE ('sys.window1, sys.window2,
   sys.window3, sys.windowgroup1, sys.windowgroup2');
END;
/
```

Note that if a window group name is specified, then the window group will be disabled, but the windows that are members of the window group, will not be disabled. A job, other than a running job, that has the window group as its schedule will not run because the window group is disabled. However, if the job had one of the window group members as its schedule, it would still run.

Disabling a window that is already disabled does not generate an error. Disabling a window that does not exist causes an error. Also, if it is not possible to disable window2, the DISABLE call will fail. window1 will be disabled but window2, window3, windowgroup1, and windowgroup2 will not be disabled.

When a window is disabled, an entry is made in the window log.

Enabling Windows

You enable a window using the ENABLE procedure. An enabled window is one that can be opened. Windows are, by default, created enabled. When a window is enabled using the ENABLE procedure, a validity check is performed and only if this is successful will the window be enabled. When a window is enabled, it is logged in the window log table. Because the ENABLE procedure is used for several Scheduler objects, when enabling windows, they must be preceded by SYS.

You can enable several windows in one call by providing a comma-delimited list of window names or window group names to the ENABLE procedure call. For example, the following statement enables both windows and window groups:

```
BEGIN
DBMS_SCHEDULER.ENABLE ('sys.window1, sys.window2,
   sys.window3, sys.windowgroup1, sys.windowgroup2');
END;
/
```

Note that if a window group name is specified, then the window group will be enabled, but the windows that are members of the window group, will not be enabled.

Note that if it is not possible to enable window2, then the ENABLE call will fail and the error returned will list the name of the window or window group that failed. window1 will be enabled but window2, window3, windowgroup1, windowgroup2 will not be enabled.

Overlapping Windows

Although Oracle does not recommend it, windows can overlap. Because only one window can be active at one time, the following rules are used to determine which window will be active when windows overlap:

- If windows of the same priority overlap, the window that is active will stay open. However, if the overlap is with a window of higher priority, the lower priority window will close and the window with the higher priority will open.
- If at the end of a window there are multiple windows defined, the window that has the highest percentage of time remaining will open.
- An open window that is dropped will be automatically closed. At that point, the previous rule applies.

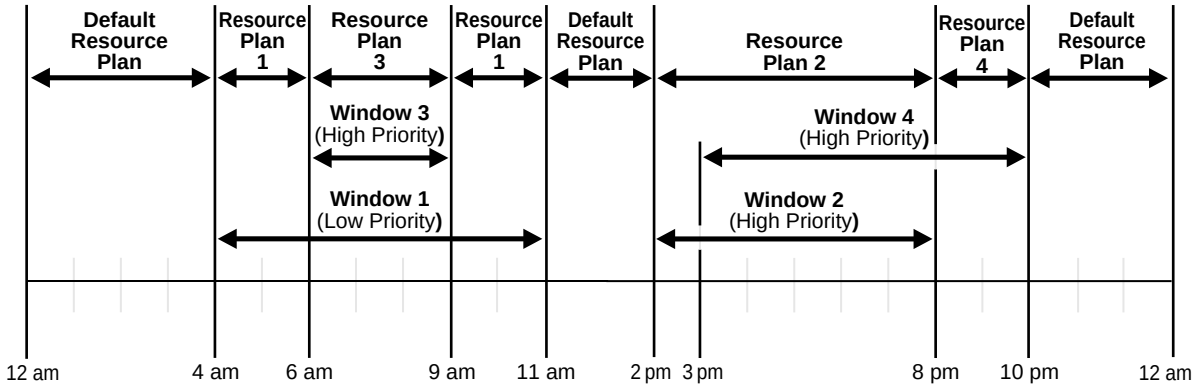
Changing Resource Plans

For the Scheduler to successfully change resource plans, you must ensure that Resource Manager is active. To verify that it is, use the V$RSRC_PLAN view. If Resource Manager is not running, you need to set the RESOURCE_MANAGER_PLAN initialization parameter in the `init.ora` file or issue an `ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = my_plan` statement. Either option will activate Resource Manager and will set the default resource plan. If the Scheduler cannot switch resource plans, the appropriate window will still open and jobs that have that window as their schedule will be picked up.

Examples of Overlapping Windows

Figure 27–1 illustrates a typical example of how windows, resource plans, and priorities might be determined for a 24 hour schedule. In the following two examples, assume that Window1 has been associated with Resource Plan1, Window2 with Resource Plan2, and so on.

Figure 27–1 Windows and Resource Plans (Example 1)



In Figure 27–1, the following occurs:

- From 12AM to 4AM

  No windows are open, so a default resource plan is in effect.

- From 4AM to 6AM

  Window1 has been assigned a low priority, but it opens because there are no high priority windows. Therefore, Resource Plan 1 is in effect.

- From 6AM to 9AM

Window3 will open because it has a higher priority than Window1, so Resource Plan 3 is in effect.

- From 9AM to 11AM

  Even though Window1 was closed at 6AM because of a higher priority window opening, at 9AM, this higher priority window is closed and Window1 still has two hours remaining on its original schedule. It will be reopened for these remaining two hours and resource plan will be in effect.

- From 11AM to 2PM

  A default resource plan is in effect because no windows are open.

- From 2PM to 3PM

  Window2 will open so Resource Plan 2 is in effect.

- From 3PM to 8PM

  Window4 is of the same priority as Window2, so it will not interrupt Window2. Therefore, Resource Plan 2 is in effect.

- From 8PM to 10PM

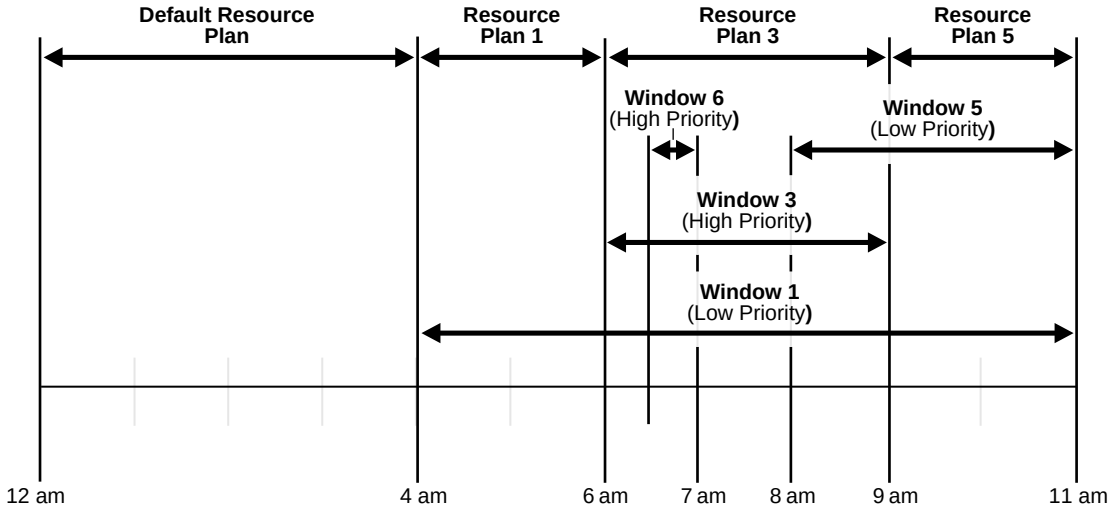
  Window4 will open so Resource Plan 4 is in effect.

- From 10PM to 12AM

  A default resource plan is in effect because no windows are open.

Figure 27–2 illustrates another example of how windows, resource plans, and priorities might be determined for a 24 hour schedule.

Figure 27–2 Windows and Resource Plans (Example 2)



In Figure 27–2, the following occurs:

- From 12AM to 4AM

  A default resource plan is in effect.

- From 4AM to 6AM

  Window1 has been assigned a low priority, but it opens because there are no high priority windows, so Resource Plan 1 is in effect.

- From 6AM to 9AM

  Window3 will open because it has a higher priority than Window1. Note that Window6 does not open because another high priority window is already in effect.

- From 9AM to 11AM

  At 9AM, Window5 or Window1 are the two possibilities. They both have low priorities, so the choice is made based on which has a greater percentage of its duration remaining. Window5 has a larger percentage of time remaining compared to the total duration than Window1. Even if Window1 were to extend to, say, 11:30AM, Window5 would have 2/3 * 100% of its duration remaining, while Window1 would have only 2.5/7 * 100%, which is smaller. Thus, Resource Plan 5 will be in effect.

Window Logging

Window activity is logged in the *_SCHEDULER_WINDOW_LOG views. See "Window Logs" on page 28-15 for examples of window logging.

Administering Window Groups

A window group is a named collection of windows. Window groups reside in the SYS schema. This section introduces you to basic window group tasks, and discusses the following topics:

- Window Group Tasks and Their Procedures
- Creating Window Groups
- Dropping Window Groups
- Adding a Member to a Window Group
- Dropping a Member from a Window Group
- Enabling a Window Group
- Disabling a Window Group

Window Group Tasks and Their Procedures

Table 27–6 illustrates common window group tasks and the procedures you use to handle them.

Table 27–6 Window Group Tasks and Their Procedures

Task	Procedure	Privilege Needed
Create a window group	CREATE_WINDOW_GROUP	MANAGE SCHEDULER
Drop a window group	DROP_WINDOW_GROUP	MANAGE SCHEDULER
Add a member to a window group	ADD_WINDOW_GROUP_MEMBER	MANAGE SCHEDULER
Drop a member to a window group	REMOVE_WINDOW_GROUP_MEMBER	MANAGE SCHEDULER
Enabling a window group	ENABLE	MANAGE SCHEDULER
Disabling a window group	DISABLE	MANAGE SCHEDULER

See "How to Manage Scheduler Privileges" on page 28-16 for further information regarding privileges.

Creating Window Groups

You create a window group by using the CREATE_WINDOW_GROUP procedure. Only a window can be a member of a window group. You can specify the windows that will be members of the group when you are creating the group, or you can add them later using the ADD_WINDOW_GROUP_MEMBER procedure. A window group cannot be a member of another window group. You can, however, create a window group that has no members.

If you create a window group and you specify a window that does not exist as its member, an error is generated and the window group is not created.

Window groups are created in the SYS schema. Window groups, like windows, are created with access to PUBLIC, therefore, no privileges are required to access window groups.

As an example, the following statement creates a window group called my_window_group1:

```
BEGIN
DBMS_SCHEDULER.CREATE_WINDOW_GROUP ('my_window_group1');
END;
/
```

Then, you could add a window (my_window1) to my_window_group1 by issuing the following statement:

```
BEGIN
DBMS_SCHEDULER.ADD_WINDOW_GROUP_MEMBER (
   window_name  =>  'my_window_group1',
   window_list  =>  'my_window1');
END;
/
```

Dropping Window Groups

You drop a window group by using the DROP_WINDOW_GROUP procedure. This call will drop the window group but not the windows that are members of this window group. If you want to drop all the windows that are members of this group but not the window group itself, you can use the DROP_WINDOW procedure and provide the name of the window group to the call.

You can also drop several window groups in one call by providing a comma-delimited list of window group names to the DROP_WINDOW_GROUP procedure call. For example, the following statement drops three window groups:

```
BEGIN
DBMS_SCHEDULER.DROP_WINDOW_GROUP('windowgroup1, windowgroup2, windowgroup3');
END;
/
```

Dropping a window group that does not exist will cause an error stating that the window group does not exist and the error mess will contain the name of the window group that failed. Note that if it is not possible to drop `windowgroup2` in the preceding example, then the `DROP_WINDOW_GROUP` call will fail. `windowgroup1` will be dropped but `windowgroup2` and `windowgroup3` will not be dropped.

If there are jobs that have the window group as their schedule, you will not be able to drop the window group unless you set `force` to `TRUE` in the procedure call. By default, `force` is set to `FALSE`. When a window group is dropped, those jobs that have the window group as their schedule will be disabled.

Running jobs that have the window group as their schedule are allowed to continue, even if the `stop_on_window_close` flag was set to `TRUE` when the job was created.

If a member of the window group that is being dropped is open, the window group can still be dropped.

See *PL/SQL Packages and Types Reference* for detailed information about adding and dropping window groups.

Adding a Member to a Window Group

You add a member to a window group by using the `ADD_WINDOW_GROUP_MEMBER` procedure. Only when a window opens will the Scheduler check whether there are any jobs whose schedule is that window or a window group of which this window is a member. Based on priority and resource availability, the Scheduler will then execute the jobs.

If a window is already open, and a new job is created that points to that window, it will not be started until the next time the window opens. If an already open window is added to a window group, the Scheduler will not pick up jobs that point to this window group until the next window in the window group opens.

You can add several members to a window group in one call, by specifying a comma-delimited list of windows. For example, the following statement adds three windows:

```
BEGIN
```

```
DBMS_SCHEDULER.ADD_WINDOW_GROUP_MEMBER ('window_group1',
   'window1, window2, window3');
END;
/
```

If any of the windows specified is either invalid or does not exist, the call fails.

Note that a window group cannot be a member of another window group.

Dropping a Member from a Window Group

You can drop a window from a window group by using the REMOVE_WINDOW_GROUP_MEMBER procedure. Jobs with the `stop_on_window_close` flag set will only be stopped when a window closes. Dropping an open window from a window group has no impact on this.

You can remove several members from a window group in one call by specifying a comma-delimited list of windows. For example, the following statement drops three windows:

```
BEGIN
DBMS_SCHEDULER.REMOVE_WINDOW_GROUP_MEMBER('window_group1', 'window1, window2,
   window3');
END;
/
```

If any of the windows specified is either invalid or does not exist, the call fails.

Enabling a Window Group

You enable a window group using the ENABLE procedure. By default, window groups are created ENABLED. You can enable several window groups in one call by providing a comma-delimited list of window group names to the ENABLE procedure call. For example, consider the following statement:

```
BEGIN
DBMS_SCHEDULER.ENABLE('sys.windowgroup1', 'sys.windowgroup2, sys.windowgroup3');
END;
/
```

In this example, the window groups will be enabled but the windows that are members of the window groups will not be enabled.

Note that if it is not possible to enable `windowgroup2`, then the ENABLE call will fail. `windowgroup1` will be enabled but `windowgroup2` and `windowgroup3` will

not be enabled. The error message in this case will list the name of the window group that failed.

Enabling a window group that is already enabled does not generate an error.

Disabling a Window Group

You disable a window group using the DISABLE procedure. This means that jobs with the window group as a schedule will not run even if the member windows open, however, the metadata of the window group is still there, so it can be reenabled. Note that the members of the window group will still open.

If a member of the window group that you are trying to disable is open, then an error occurs unless the force option is set to TRUE in the procedure call. If the window group is disabled, the open window will be not closed or disabled. It will be allowed to continue to its end.

If there are jobs that have the window group as their schedule, you will not be able to disable the window group unless you set force to TRUE in the procedure call. By default, force is set to FALSE. When the window group is disabled, those jobs that have the window group as their schedule will not be disabled.

You can also disable several window groups in one call by providing a comma-delimited list of window group names to the DISABLE procedure call. For example, the following statement disables three window groups:

```
BEGIN
DBMS_SCHEDULER.DISABLE('sys.windowgroup1, sys.windowgroup2, sys.windowgroup3');
END;
/
```

Note that, in this example, the window group will be disabled, but the windows that are members of the window group will not be disabled.

Note that if it is not possible to disable windowgroup2, then the DISABLE call will fail. windowgroup1 will be disabled but windowgroup2 and windowgroup3 will not be disabled. The error message in this case will list the name of the window group that failed. Disabling a window group that is already disabled will not generate an error.

Allocating Resources Among Jobs

It is not practical to manage resource allocation at an individual job level, therefore, the Scheduler uses the concept of job classes to manage resource allocation among

jobs. In addition to job classes, the Scheduler uses the Resource Manager to manage resource allocation among jobs.

Allocating Resources Among Jobs Using Resource Manager

Resource Manager is the database feature that controls how resources are allocated in the database. It not only controls asynchronous sessions like jobs but also synchronous sessions like user sessions. It groups all "units of work" in the database into resource consumer groups and uses a resource plan to specify how the resources will be allocated among the various groups. See Chapter 24, "Using the Database Resource Manager" for more information about what resources are controlled by resource manager.

For jobs, resource allocation is specified by mapping a job class to a consumer group. The consumer group that a job class maps to can be specified when creating a job class. If no resource consumer group is specified when a job class is created, the job class will map to the default consumer group. Because the consumer group is an attribute of a job class, it can be changed after the job class has been created using the `SET_ATTRIBUTE` procedure.

Because all jobs must belong to a job class and a job class is always associated with a resource consumer group, resource manager will always be able to properly allocate resources among jobs.

The Scheduler tries to limit the number of jobs that are running simultaneously so that at least some jobs can complete rather than running a lot of jobs concurrently but without enough resources for any of them to complete. Therefore, the job coordinator only starts jobs if there are enough resources available to run them.

The Scheduler and Resource Manager are tightly integrated. The job coordinator obtains database resource availability from Resource Manager. Based on that information, the coordinator determines how many jobs to start. It will only start jobs from those job classes that will have enough resources to run. The coordinator will keep starting jobs in a particular job class that maps to a consumer group till Resource Manager determines that the maximum resource allocated for that consumer group has been reached. This means that it is possible that there will be jobs in the job table that are ready to run but will not be picked up by the job coordinator because there are no resources to run them. Therefore, there is no guarantee that a job will run at the exact time it was scheduled to. The coordinator picks up jobs from the job table on the basis of which consumer groups still have resources available.

Even when jobs are running, Resource Manager will continue to manage the amount of CPU cycles that are assigned to each running job based on the specified

resource plan. Keep in mind that Resource Manager can only manage database processes. The active management of CPU cycles does not apply to jobs of type `executable`.

In a database, only one resource plan can be in effect at one time. It is possible to manually switch the resource plan that is active on a system using the `ALTER SYSTEM` statement. In special scenarios, a database administrator might want to run a specific resource plan without the Scheduler switching to its Scheduler resource plans associated with windows. To do this, use the `ALTER SYSTEM SET RESOURCE_MANAGER_PLAN` statement with the `force` option.

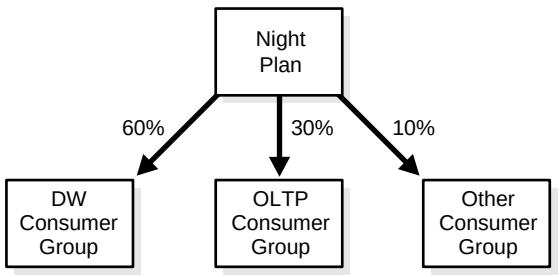
Note: You must ensure that Resource Manager is active. Otherwise, the Scheduler will not be able to switch resource plans.To verify that it is, use the `V$RSRC_PLAN` view. If Resource Manager is not running, you need to set the `RESOURCE_MANAGER_PLAN` initialization parameter in the `init.ora` file or issue an `ALTER SYSTEM SET RESOURCE_MANAGER_PLAN =` *`my_plan`* statement.

In a RAC environment, the same resource plan will be in effect on every database instance.

Example of Resource Allocation for Jobs

The following example can help to understand how resources are allocated for jobs. Assume there are three job classes: `JC1`, which maps to consumer group `DW`; `JC2`, which maps to consumer group `OLTP`; and `JC3`, which maps to the default consumer group. Figure 27–3 offers a simple graphical illustration of this scenario.

Figure 27–3 Sample Resource Plan



This resource plan clearly gives priority to jobs that are part of job class JC1. Consumer group DW gets 60% of the resources, thus jobs that belong to job class JC1 will get 60% of the resources. Consumer group OLTP has 30% of the resources, which implies that jobs in job class JC2 will get 30% of the resources. The consumer group Other specifies that all other consumer groups will be getting 10% of the resources. This means that all jobs that belong in job class JC3 will share 10% of the resources and can get a maximum of 10% of the resources.

Note that a resource plan that specifies 100% of the resources to the consumer group Other is not the same as a resource plan that equally splits the resources among all consumer groups. In the first case, there is no active resource management. In the second case, Resource Manager will actively try to allocate resources equally among all the consumer groups. As an example, (CG DW 50% and CG OLTP 50%) do not equal Other 100%.

28

Administering the Scheduler

Oracle Database provides database scheduling capabilities through the Scheduler. This chapter describes managing a Scheduler environment, including step-by-step instructions for configuring, administering, and monitoring. This chapter contains the following sections:

- Configuring the Scheduler
- Monitoring and Managing the Scheduler
- Import/Export and the Scheduler
- Examples of Using the Scheduler

> **Note:** This chapter discusses the use of the Oracle-supplied `DBMS_SCHEDULER` package to administer scheduling capabilities. You can also use Oracle Enterprise Manager (EM) as an easy-to-use graphical interface for many of the same capabilities.
>
> See the *PL/SQL Packages and Types Reference* for `DBMS_SCHEDULER` syntax and the Oracle Enterprise Manager documentation set for more information regarding EM.

Configuring the Scheduler

The following tasks are necessary when configuring the Scheduler:

- Task 1: Setting Scheduler Privileges
- Task 2: Configuring the Scheduler Environment

Task 1: Setting Scheduler Privileges

You should have the SCHEDULER_ADMIN role to administer the Scheduler. Typically, database administrators will already have this role with the ADMIN option as part of the DBA (or equivalent) role. You can grant this role to another administrator by issuing the following statement:

```
GRANT SCHEDULER_ADMIN TO username;
```

Because the SCHEDULER_ADMIN role is a powerful role allowing a grantee to execute code as any user, you should consider granting individual Scheduler system privileges instead. Object and system privileges are granted using regular SQL grant syntax. An example is if the database administrator issues the following statement:

```
GRANT CREATE JOB TO scott;
```

After this statement is executed, scott can create jobs, schedules, or programs in his schema. Another example is if the database administrator issues the following statement:

```
GRANT MANAGE SCHEDULER TO adam;
```

After this statement is executed, adam can create, alter, or drop windows, job classes, or window groups. He will also be able to set and retrieve Scheduler attributes and purge Scheduler logs.

See "How to Manage Scheduler Privileges" on page 28-16 for more information regarding privileges.

Task 2: Configuring the Scheduler Environment

This section discusses the following tasks:

- Task 2A: Creating Job Classes
- Task 2B: Creating Windows
- Task 2C: Creating Resource Plans
- Task 2D: Creating Window Groups
- Task 2E: Setting Scheduler Attributes

Task 2A: Creating Job Classes

To create job classes, use the CREATE_JOB_CLASS procedure. The following statement illustrates an example of creating a job class:

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB_CLASS (
   job_class_name             => 'my_jobclass1',
   resource_consumer_group    => 'my_res_group1',
   comments                   => 'This is my first job class.');
END;
/
```

This statement creates a job class called `my_jobclass1` with attributes such as a resource consumer group of `my_res_group1`. To verify the job class contents, issue the following statement:

```
SELECT * FROM DBA_SCHEDULER_JOB_CLASSES;

JOB_CLASS_NAME         RESOURCE_CONSU   SERVICE    LOGGING_LEV  LOG_HISTORY    COMMENTS
-----------------      --------------   -------    -----------  -----------    --------
DEFAULT_JOB_CLASS                                       RUNS                  The default
AUTO_TASKS_JOB_CLASS   AUTO_TASK_CON                    RUNS                  System maintenance
FINANCE_JOBS           FINANCE_GROUP                    RUNS
MY_JOBCLASS1           MY_RES_GROUP1                    RUNS                  My first job class
MY_CLASS1                             my_service1       RUNS                  My second job class
5 rows selected.
```

Note that job classes are created in the SYS schema.

> **See Also:** *PL/SQL Packages and Types Reference* for CREATE_JOB_CLASS syntax, "Creating Job Classes" on page 27-24 for further information on job classes, and "Examples of Creating Job Classes" on page 28-24 for more examples of creating job classes

Task 2B: Creating Windows

To create windows, use the CREATE_WINDOW procedure. The following statement illustrates an example of creating a window:

```
BEGIN
DBMS_SCHEDULER.CREATE_WINDOW (
   window_name     =>  'my_window1',
   resource_plan   =>  'my_resourceplan1',
   start_date      =>  '15-APR-03 01.00.00 AM Europe/Lisbon',
   repeat_interval =>  'FREQ=DAILY',
   end_date        =>  '15-SEP-04 01.00.00 AM Europe/Lisbon',
   duration        =>  interval '50' minute,
   window_priority =>  'HIGH',
```

```
   comments        =>  'This is my first window.');
END;
/
```

This statement creates a window called my_window1 with attributes such as a resource plan of my_resourceplan1. To verify the window contents, query the view DBA_SCHEDULER_WINDOWS. As an example, issue the following statement:

```
SELECT WINDOW_NAME, RESOURCE_PLAN, DURATION, REPEAT_INTERVAL
FROM DBA_SCHEDULER_WINDOWS;

WINDOW_NAME    RESOURCE_PLAN     DURATION         REPEAT_INTERVAL
-----------    -------------     -------------    ---------------
MY_WINDOW1     MY_RESOURCEPLAN1  +000 00:50:00    FREQ=DAILY
```

> **See Also:** *PL/SQL Packages and Types Reference* for CREATE_WINDOW syntax, "Creating Windows" on page 27-27 for further information on windows, and "Examples of Creating Windows" on page 28-26 for more examples of creating job classes

Task 2C: Creating Resource Plans

To create resource plans, use the CREATE_SIMPLE_PLAN procedure. This procedure enables you to create consumer groups and allocate resources to them by executing a single statement. If you do not create a resource plan, the Scheduler uses a default resource plan called INTERNAL_PLAN.

The following statement illustrates an example of using this procedure to create a resource plan called my_simple_plan1:

```
BEGIN
DBMS_RESOURCE_MANAGER.CREATE_SIMPLE_PLAN (
   simple_plan       => 'my_simple_plan1',
   consumer_group1   => 'my_group1',
   group1_cpu        => 80,
   consumer_group2   => 'my_group2',
   group2_cpu        => 20);
END;
/
```

This statement creates a resource plan called my_simple_plan1. To verify the resource plan contents, query the view DBA_RSRC_PLANS. An example is the following statement:

```
SELECT PLAN, STATUS FROM DBA_RSRC_PLANS;
```

```
PLAN                           STATUS
------------------------------ -------------------------
SYSTEM_PLAN                    ACTIVE
INTERNAL_QUIESCE               ACTIVE
INTERNAL_PLAN                  ACTIVE
MY_SIMPLE_PLAN1                ACTIVE
```

See Also: "Allocating Resources Among Jobs" on page 27-41 for further information on resource plans

Task 2D: Creating Window Groups

To create window groups, use the CREATE_WINDOW_GROUP and ADD_WINDOW_GROUP_MEMBER procedures. The following statements illustrate an example of using these procedures:

```
BEGIN
DBMS_SCHEDULER.CREATE_WINDOW_GROUP (
   group_name        =>  'my_window_group1',
   comments          =>  'This is my first window group.');

DBMS_SCHEDULER.ADD_WINDOW_GROUP_MEMBER (
   group_name        =>  'my_window_group1',
   window_list       =>  'my_window1, my_window2');

DBMS_SCHEDULER.ADD_WINDOW_GROUP_MEMBER (
   group_name        =>  'my_window_group1',
   window_list       =>  'my_window3');
END;
/
```

These statements assume that you have already created my_window2 and my_window3. You can do this with the CREATE_WINDOW procedure.

These statements create a window group called my_window_group1 and then add my_window1, my_window2, and my_window3 to it. To verify the window group contents, issue the following statements:

```
SELECT * FROM DBA_SCHEDULER_WINDOW_GROUPS;

WINDOW_GROUP_NAME   ENABLED  NUMBER_OF_WINDOWS   COMMENTS
-----------------   -------  -----------------   --------------------
MY_WINDOW_GROUP1    TRUE                     3   This is my first window group.
```

```
SELECT * FROM DBA_SCHEDULER_WINGROUP_MEMBERS;

WINDOW_GROUP_NAME              WINDOW_NAME
------------------------------ ---------------
MY_WINDOW_GROUP1               MY_WINDOW1
MY_WINDOW_GROUP1               MY_WINDOW2
MY_WINDOW_GROUP1               MY_WINDOW3
```

> **See Also:** *PL/SQL Packages and Types Reference* for CREATE_WINDOW_GROUP syntax, "Administering Window Groups" on page 27-37 for further information on window groups, and "Example of Creating Window Groups" on page 28-27 for more detailed examples of creating window groups

Task 2E: Setting Scheduler Attributes

There are several Scheduler attributes that control the behavior of the Scheduler. They are `default_timezone`, `log_history`, and `max_job_slave_processes`. It is crucial that you set the `default_timezone` attribute because it impacts the behavior of repeating jobs and windows. The other two have defaults, but you may want to change the default settings. The values of these attributes can be set by using the SET_SCHEDULER_ATTRIBUTE procedure. Setting these attributes requires the MANAGE SCHEDULER privilege. Attributes that can be set are:

- `default_timezone`

  Repeating jobs and windows that use the calendaring syntax need to know which time zone to use for their repeat intervals. This is normally retrieved from `start_date`, but if no `start_date` is provided (which is not uncommon), the time zone is retrieved from this Scheduler attribute. To make sure that daylight savings adjustments are followed, it is strongly recommended to set this attribute to a region name instead of an absolute time zone offset. For example, if your database resides in Miami, issue the following statement:

  ```
  DBMS_SCHEDULER.SET_SCHEDULER_ATTRIBUTE('default_timezone','US/Eastern');
  ```

  If you do not set this attribute, the default time zone for repeating jobs and windows will be the absolute offset retrieved from SYSTIMESTAMP (the time zone of the OS environment of the database), which means that repeating jobs and windows that do not have their `start_date` set will not follow daylight savings adjustments.

- `log_history`

This enables you to control the amount of logging the Scheduler performs. To prevent the job log and the window log from growing indiscriminately, the Scheduler has an attribute that specifies how much history (in days) to keep. Once a day, the Scheduler automatically purges all log entries from both the job log as well as the window log that are older than the specified history. The default is 30 days.

You can change the default by using the SET_SCHEDULER_ATTRIBUTE procedure. For example, to change it to 90 days, issue the following statement:

```
DBMS_SCHEDULER.SET_SCHEDULER_ATTRIBUTE('log_history','90');
```

The range of valid values is 1 through 999.

- max_job_slave_processes

  This enables you to set a maximum number of slave processes for a particular system configuration and load. Even though the Scheduler automatically determines what the optimum number of slave processes is for a given system configuration and load, you still might want to set a fixed limit on the Scheduler. If this is the case, you can set this attribute. The default value is NULL, and the valid range is 1-999.

  Although the number set by max_job_slave_processes is a real maximum, it does not mean the Scheduler will start the specified number of slaves. For example, even though this attribute is set to 10, the Scheduler might still determine that is should not start more than 3 slave processes. However, if it wants to start 15, but it is set to 10, it will not start more than 10.

  See *PL/SQL Packages and Types Reference* for detailed information about the SET_SCHEDULER_ATTRIBUTE procedure.

Monitoring and Managing the Scheduler

The following sections discuss how to monitor and manage the Scheduler:

- How to View Scheduler Information
- How to View the Currently Active Window and Resource Plan
- How to View Scheduler Privileges
- How to Find Information About Currently Running Jobs
- How the Job Coordinator Works
- How to Monitor and Manage Window and Job Logs

- How to Manage Scheduler Privileges
- How to Drop a Job
- How to Drop a Running Job
- Why Does a Job Not Run?
- How to Change Job Priorities
- How the Scheduler Guarantees Availability
- How to Handle Scheduler Security
- How to Manage the Scheduler in a RAC Environment

How to View Scheduler Information

You can check Scheduler information by using many views. An example is the following, which shows information for completed instances of `my_job1`:

```
SELECT JOB_NAME, STATUS, ERROR#
FROM DBA_SCHEDULER_JOB_RUN_DETAILS WHERE JOB_NAME = 'MY_JOB1';

JOB_NAME     STATUS           ERROR#
--------     --------------   ------
MY_JOB1      FAILURE           20000
```

Table 28–1 contains views associated with the Scheduler. The `*_SCHEDULER_JOBS`, `*_SCHEDULER_SCHEDULES`, `*_SCHEDULER_PROGRAMS`, `*_SCHEDULER_RUNNING_JOBS`, `*_SCHEDULER_JOB_LOG`, `*_SCHEDULER_JOB_RUN_DETAILS` views are particularly useful for managing jobs. See *Oracle Database Reference* for details regarding Scheduler views.

Table 28–1 Scheduler Views

View	Description
`*_SCHEDULER_SCHEDULES`	These views show all schedules.
`*_SCHEDULER_PROGRAMS`	These views show all programs.
`*_SCHEDULER_PROGRAM_ARGUMENTS`	These views show all arguments registered with all programs as well as the default values if they exist.
`*_SCHEDULER_JOBS`	These views show all jobs, enabled as well as disabled.
`*_SCHEDULER_GLOBAL_ATTRIBUTE`	These views show the current values of Scheduler attributes.

Table 28–1 (Cont.) Scheduler Views

View	Description
*_SCHEDULER_JOB_ARGUMENTS	These views show all arguments for all jobs, assigned and unassigned.
*_SCHEDULER_JOB_CLASSES	These views show all job classes.
*_SCHEDULER_WINDOWS	These views show all windows.
*_SCHEDULER_JOB_RUN_DETAILS	These views show all completed (failed or successful) job runs.
*_SCHEDULER_WINDOW_GROUPS	These views show all window groups.
*_SCHEDULER_WINGROUP_MEMBERS	These views show the members of all window groups, one row for each group member.
*_SCHEDULER_RUNNING_JOBS	These views show state information on all jobs that are currently being run.

How to View the Currently Active Window and Resource Plan

You can view the currently active window and the plan associated with it by issuing the following statement:

```
SELECT WINDOW_NAME, RESOURCE_PLAN FROM DBA_SCHEDULER_WINDOWS
WHERE ACTIVE='TRUE';

WINDOW_NAME                    RESOURCE_PLAN
------------------------------ ------------------------
MY_WINDOW10                    MY_RESOURCEPLAN1
```

If there is no window active, you can view the active resource plan by issuing the following statement:

```
SELECT * FROM V$RSRC_PLAN;
```

How to View Scheduler Privileges

You must have the MANAGE SCHEDULER privilege to administer the Scheduler. Typically, database administrators have this privilege with the ADMIN option as part of the DBA (or equivalent) role. You can check your current system privileges by issuing the following statement:

```
SELECT * FROM SESSION_PRIVS;
```

If you do not have sufficient privileges, see "Task 1: Setting Scheduler Privileges" on page 28-2, "How to Manage Scheduler Privileges" on page 28-16, and *Oracle Database Security Guide*.

How to Find Information About Currently Running Jobs

You can check job state by issuing the following statement:

```
SELECT JOB_NAME, STATE FROM DBA_SCHEDULER_JOBS
WHERE JOB_NAME = 'MY_EMP_JOB1';

JOB_NAME                       STATE
------------------------------ ---------
MY_EMP_JOB1                    DISABLED
```

In this case, you could enable the job using the ENABLE procedure. Table 28–2 shows the valid values for job state.

Table 28–2 Job States

Job State	Description
`disabled`	The job is disabled.
`scheduled`	The job is scheduled to be executed.
`running`	The job is currently running.
`completed`	The job has completed, and is not scheduled to run again.
`broken`	The job is broken.
`failed`	The job was scheduled to run once and failed.
`retry scheduled`	The job has failed at least once and a retry has been scheduled to be executed.
`succeeded`	The job was scheduled to run once and completed successfully.

You can check whether the progress of currently running jobs by issuing the following statement:

```
SELECT * FROM DBA_SCHEDULER_RUNNING_JOBS;
```

Note that, for the column CPU_USED, the initialization parameter RESOURCE_LIMIT must be set to true.

You can check whether the job coordinator is running by searching for a process of the form `cjqNNN`.

See Also: *Oracle Database Reference* for details regarding the *_SCHEDULER_RUNNING_JOBS and DBA_SCHEDULER_JOBS views

How the Job Coordinator Works

The job coordinator background process is automatically started and stopped on an as-needed basis. By default, the coordinator will not be up and running, but the database does monitor whether there are any jobs to be executed, or windows to be opened in the near future. If so it will start the coordinator.

As long as there are jobs or windows running, the coordinator continues to be up. Once there has been a certain period of Scheduler inactivity and there are no jobs or windows scheduled in the near future, the coordinator will automatically be stopped.

Job Coordinator and Real Application Clusters

Each RAC instance has its own job coordinator. The database monitoring checks that determine whether or not to start the job coordinator do take the service affinity of jobs into account. For example, if there is only one job scheduled in the near future and the job class to which this job belongs has service affinity for only two out of the four RAC instances, only the job coordinators for those two instances will be started.

Using DBMS_SCHEDULER and DBMS_JOB at the Same Time

Even though Oracle recommends you switch from DBMS_JOB to DBMS_SCHEDULER, DBMS_JOB is still supported for backward compatibility. Both Scheduler packages share the same job coordinator, but DBMS_JOB does not have the auto start and stop functionality. Instead, the job coordinator is controlled by the JOB_QUEUE_PROCESSES initialization parameter. When JOB_QUEUE_PROCESSES is set to 0, the coordinator is turned off and when it has a non-zero value it is turned on.

The JOB_QUEUE_PROCESSES initialization parameter is only used for DBMS_JOB. When this parameter is set to a non-zero value, auto start and stop no longer apply because the coordinator will always be up and running. In this case, the coordinator will take care of execution of both DBMS_SCHEDULER and DBMS_JOB jobs.

If the initialization parameter is set to 0, or if it is not set at all, no DBMS_JOB jobs will be run, however, the auto start and stop feature will be used for all DBMS_SCHEDULER jobs and windows. If there is a DBMS_SCHEDULER job to be executed, the coordinator will be started and the job will be executed. However, DBMS_JOB jobs still will not be run.

Scheduler Attribute max_job_slave_processes

The initialization parameter JOB_QUEUE_PROCESSES only applies to DBMS_JOB. When DBMS_SCHEDULER is used, the coordinator will automatically determine how many job slaves to start based on CPU load and the number of outstanding jobs. In special scenarios a dba can still limit the maximum number of slaves to be started by the coordinator by setting the MAX_JOB_SLAVE_PROCESSES with the DBMS_SCHEDULER.SET_SCHEDULER_ATTRIBUTE procedure.

How to Monitor and Manage Window and Job Logs

Logs have a new entry for each event that occurs so you can track historical information. This is different from a queue, where you only want to track the latest status for an item. There are logs for jobs, job runs, and windows.

Job activity is logged in the *_SCHEDULER_JOB_LOG views. Altering a job is logged with a status of UPDATE. Dropping a job is logged in these views with a status of DROP.

Job Logs

A job log has an entry for each time you create or drop a job. To see the contents of the job log, query the DBA_SCHEDULER_JOB_LOG view. An example is the following statement, which shows what happened for past job runs:

```
SELECT JOB_NAME, OPERATION, OWNER FROM DBA_SCHEDULER_JOB_LOG;

JOB_NAME        OPERATION     OWNER
--------        ---------     -----
MY_JOB13        CREATE        SYS
MY_JOB14        CREATE        OE
MY_NEW_JOB3     CREATE        SYS
MY_JOB1         CREATE        SYS
MY_TEST_JOB1    CREATE        SYS
MY_TEST_JOB2    CREATE        SYS
MY_TEST_JOB2    CREATE        OE
MY_JOB11        CREATE        OE
MY_TEST_JOB4    CREATE        OE
MY_TEST_JOB5    CREATE        OE
MY_JOB12        CREATE        OE
MY_NEW_JOB3     ENABLE        SYS
MY_EMP_JOB1     UPDATE        SYS
MY_JOB1         CREATE        SCOTT
MY_EMP_JOB1     UPDATE        SYS
MY_EMP_JOB      CREATE        SYS
```

```
MY_EMP_JOB1     CREATE        SYS
MY_JOB14        RUN           OE
MY_JOB14        RETRY_RUN     OE
MY_JOB14        RETRY_RUN     OE
MY_JOB14        RETRY_RUN     OE
MY_JOB14        RETRY_RUN     OE
MY_JOB14        BROKEN        OE
MY_NEW_JOB1     CREATE        SYS
MY_JOB14        DROP          OE
MY_NEW_JOB2     CREATE        SYS
```

Job Run Details

To further analyze each job run, for example, why it failed, or what the actual start time was, or how long the job ran, query the DBA_SCHEDULER_JOB_RUN_DETAILS view. As an example, the following statement illustrates the status for my_job14:

```
SELECT JOB_NAME, STATUS FROM DBA_SCHEDULER_JOB_RUN_DETAILS
WHERE JOB_NAME = 'MY_JOB14';

JOB_NAME                       STATUS
------------------------------ ------------------------------
MY_JOB14                       FAILURE
MY_JOB14                       FAILURE
MY_JOB14                       FAILURE
MY_JOB14                       FAILURE
MY_JOB14                       FAILURE
```

For every row in SCHEDULER_JOB_LOG that is of operation RUN or RETRY_RUN, there will be a corresponding row in *_JOB_RUN_DETAILS view with the same LOG_ID. LOG_DATE contains the timestamp of the entry, so sorting by LOG_DATE should give you a chronological picture of the life of a job.

Controlling Job Logging

You can control the amount of logging the Scheduler performs on jobs at either a class or job level. Normally, you will want to control jobs at a class level as this offers a full audit trail. To do this, use the logging_level attribute in the CREATE_JOB_CLASS procedure.

For each new class, the creator of the class must specify what the logging level is for all jobs in that class. The three possible options are:

- DBMS_SCHEDULER.LOGGING_OFF

  No logging will be performed for any jobs in this class.

- DBMS_SCHEDULER.LOGGING_RUNS

  The Scheduler will write detailed information to the job log for all runs of each job in this class.

- DBMS_SCHEDULER.LOGGING_FULL

  In addition to recording every run of a job, the Scheduler will record all operations performed on all jobs in this class. In other words, every time a job is created, enabled, disabled, altered, and so on will be recorded in the log.

By default, only job runs are recorded. For job classes that have very short and highly frequent jobs, the overhead of recording every single run might be too much and you might choose to turn the logging off. You might, however, prefer to have a complete audit trail of everything that happened to the jobs in a specific class, in which case you need to turn on full logging for that class.

The second way of controlling the logging level is on an individual job basis. You should keep in mind, however, that the log in many cases is used as an audit trail, thus if you want a certain level of logging, the individual job creator must not be able to turn logging off. The class-specific level is, therefore, the minimum level at which job information will be logged. A job creator can only turn on more logging for an individual job, not less.

This functionality is provided for debugging purposes. For example, if the class-specific level is set to record job runs and the job-specific logging is turned off, the Scheduler will still log the runs. If, on the other hand, the job creator turns on full logging and the class-specific level is set to record runs only, all operations on this individual job will be logged. This way, an end user can test his job by turning on full logging.

To set the logging level of an individual job, you must use the SET_ATTRIBUTE procedure on that job. For example, to turn on full logging for a job called mytestjob, issue the following statement:

```
DBMS_SCHEDULER.SET_ATTRIBUTE (
   'mytestjob', 'logging_level', DBMS_SCHEDULER.LOGGING_FULL);
```

> **See Also:** *PL/SQL Packages and Types Reference* for detailed information about the CREATE_JOB_CLASS and SET_ATTRIBUTE procedures and "Task 2E: Setting Scheduler Attributes" on page 28-6

Window Logs

A window log has an entry for each time you do the following:

- create a window
- drop a window
- open a window
- close a window
- overlap windows
- disable a window
- enable a window

There are no logging levels for window logging, but every window operation will automatically be logged by the Scheduler.

For every row in a window log that is of a close operation, there will be a corresponding row in the WINDOW_DETAILS view, with the same log_id. To see the contents of the window log, query the DBA_SCHEDULER_WINDOW_LOG view. As an example, issue the following statement:

```
SELECT WINDOW_NAME, USER_NAME FROM DBA_SCHEDULER_WINDOW_LOG;

WINDOW_NAME                    USER_NAME
------------------------------ ------------------------------
MY_WINDOW10                    SYS
MY_WINDOW100                   SYS
MY_WINDOW100                   SYS
MY_WINDOW10                    SYS
MY_WINDOW100                   SYS
MY_WINDOW100                   SYS
MY_WINDOW1                     SYS
MY_WINDOW2                     SYS
MY_WINDOW3                     SYS
MY_WINDOW10                    SYS
MY_WINDOW100                   SYS
```

There is also a window details view that gives information about every window that was active and completed.

Purging Logs

To prevent job and window logs from growing indiscriminately, use the SET_SCHEDULER_ATTRIBUTE attribute to specify how much history (in days) to keep.

The Scheduler automatically purges all log entries from both the job log and the window log that are older than the history you specify. It does this once a day, and the default is 30 days, which can be changed by using the SET_SCHEDULER_ATTRIBUTE procedure. For example, to change the number of days to 90, issue the following statement:

```
DBMS_SCHEDULER.SET_SCHEDULER_ATTRIBUTE('log_history','90');
```

Some job classes are more important than others. Because of this, you can override this global history setting by using a class-specific setting. An example is if there are three classes (`class1`, `class2`, and `class3`) and you want to keep 10 days of history for the window log and `class1` and `class3`, but 30 days for `class2`. To achieve this, issue the following statements:

```
DBMS_SCHEDULER.SET_SCHEDULER_ATTRIBUTE('log_history','10');
DBMS_SCHEDULER.SET_ATTRIBUTE('class2','log_history','30');
```

You can also set the class-specific history when creating the job class.

Besides the Scheduler automatically purging the log once a day based on the log history specified, you might also want to manually purge the log. To do this, use the PURGE_LOG procedure. As an example, the following statement purges all entries from both the job and window logs:

```
DBMS_SCHEDULER.PURGE_LOG();
```

Another example is the following, which purges all entries from the jog log that are older than three days. The window log will stay as is.

```
DBMS_SCHEDULER.PURGE_LOG(log_history => 3, which_log => 'JOB_LOG');
```

The following statement purges all window log entries older than 10 days, all job log entries relating to `job1`, and all jobs in `class2` that are older than 10 days:

```
DBMS_SCHEDULER.PURGE_LOG(log_history => 10, job_name => 'job1, sys.class2');
```

How to Manage Scheduler Privileges

You should have the SCHEDULER_ADMIN role to administer the Scheduler. Typically, database administrators will already have this role with the ADMIN option as part of the DBA (or equivalent) role. You can grant this role to another administrator by issuing the following statement:

```
GRANT SCHEDULER_ADMIN TO username;
```

Because the SCHEDULER_ADMIN role is a powerful role allowing a grantee to execute code as any user, you should consider granting individual Scheduler system privileges instead. Both system and object privileges are granted using regular SQL grant syntax. An example is for the database administrator to issue the following statement:

```
GRANT CREATE JOB TO scott;
```

After this statement is executed, scott can create jobs, schedules, or programs in his schema. Another example is to grant an object privilege, as in the following statement:

```
GRANT ALTER myjob1 TO scott;
```

After this statement is executed, scott can execute, alter, or copy myjob1. See *Oracle Database SQL Reference* for system and object privilege details and *Oracle Database Security Guide* for general information.

An alternative to the SCHEDULER_ADMIN role for administering the Scheduler is to use the MANAGE SCHEDULER privilege, which is recommended for managing resources. As an example of granting this privilege to adam, issue the following statement:

```
GRANT MANAGE SCHEDULER TO adam;
```

After this statement is executed, adam can create, alter, or drop windows, job classes, or window groups. He will also be able to set and retrieve Scheduler attributes and purge Scheduler logs.

Types of Privileges and Their Descriptions

The following privileges are important when using the Scheduler.

Table 28–3 Scheduler Privileges

Privilege Name	Operations Authorized
System Privileges:	
CREATE JOB	This privilege enables you to create jobs, schedules, and programs in your own schema. You will always be able to alter and drop jobs, schedules and programs in your own schema, even if you do not have the CREATE JOB privilege. In this case, the job must have been created in your schema by another user with the CREATE ANY JOB privilege.
CREATE ANY JOB	This privilege enables you to create, alter, and drop jobs, schedules, and programs in any schema. This privilege is very powerful and should be used with care because it allows the grantee to execute code as any other user.
EXECUTE ANY PROGRAM	This privilege enables your jobs to use programs from any schema.
EXECUTE ANY CLASS	This privilege enables your jobs to run under any job class.
MANAGE SCHEDULER	This is the most important privilege for administering the Scheduler. It enables you to create, alter, and drop job classes, windows, and window groups. It also enables you to set and retrieve Scheduler attributes and purge Scheduler logs.
Object Privilege:	
EXECUTE	This privilege enables you to create a job which runs with the program or job class. It also enables you to view object attributes. It can only be granted for programs and job classes.
ALTER	This privilege enables you to alter or drop the object it is granted on. Altering includes such operations as enabling, disabling, defining or dropping program arguments, setting or resetting job argument values and running a job. For programs and jobs, this privilege enables you to view object attributes. This privilege can only be granted on jobs, programs and schedules. For other types of Scheduler objects, you can grant the MANAGE SCHEDULER system privilege. This privilege can be granted for: jobs (DROP_JOB, RUN_JOB, SET_JOB_ARGUMENT_VALUE, RESET_JOB_ARGUMENT_VALUE, SET_JOB_ANYDATA_VALUE) and (STOP_JOB without the `force` option) programs (DROP_PROGRAM, DEFINE_PROGRAM_ARGUMENT, DEFINE_ANYDATA_ARGUMENT, DEFINE_METADATA_ARGUMENT, DROP_PROGRAM_ARGUMENT, GET_ATTRIBUTE, SET_ATTRIBUTE, SET_ATTRIBUTE_NULL) schedules (DROP_SCHEDULE)

Table 28–3 (Cont.) Scheduler Privileges

Privilege Name	Operations Authorized
ALL	This privilege authorizes operations allowed by all other object attributes possible for a given object. It can be granted on jobs, programs, schedules and job classes.
SCHEDULER_ADMIN:	
All Pre-Defined Roles	The SCHEDULER_ADMIN role is created with all of the preceding system privileges (with the ADMIN option). The SCHEDULER_ADMIN role is granted to dba (with the ADMIN option).

How to Drop a Job

You can remove a job from the database by issuing a DROP_JOB statement, as in the following:

```
BEGIN
DBMS_SCHEDULER.DROP_JOB (
   job_name    =>  'my_job1');
END;
/
```

See Also: *PL/SQL Packages and Types Reference* for DROP_JOB procedure syntax

How to Drop a Running Job

You can delete a running job by issuing the DROP_JOB procedure with the `force` option. For example, the following statement forces the deletion of `my_job1`:

```
BEGIN
DBMS_SCHEDULER.DROP_JOB (
   job_name   =>  'my_job1',
   force      =>  TRUE);
END;
/
```

Note that this statement will fail if `my_job1` is running and you do not use the `force` option.

If the `force` option is specified, it will try to stop the job by using an interrupt mechanism. (which would be equivalent to calling STOP_JOB without `force` first). Alternatively, you can call STOP_JOB to first stop the job and then call DROP_JOB to drop it. If you have the MANAGE SCHEDULER privilege, you can call STOP_JOB with `force`, if the regular STOP_JOB call failed to stop the job, and then call DROP_JOB.

Why Does a Job Not Run?

A job may fail to run for several reasons. First, you should check that the job is not running by issuing the following statement:

```
SELECT JOB_NAME, STATE FROM DBA_SCHEDULER_JOBS;
```

Typical output will resemble the following:

```
JOB_NAME                       STATE
------------------------------ ---------
MY_EMP_JOB                     DISABLED
MY_EMP_JOB1                    DISABLED
MY_NEW_JOB1                    DISABLED
MY_NEW_JOB2                    DISABLED
MY_NEW_JOB3                    DISABLED
```

There are four types of jobs that are not running:

- Failed Jobs
- Broken Jobs
- Disabled Jobs
- Completed Jobs

Failed Jobs

If a job has the status of `failed` in the job table, it was scheduled to run once but the execution has failed. If the job was specified as restartable, all retries have failed.

If a job fails in the middle of execution, only the last transaction of that job is rolled back. If your job executes multiple transactions, you need to be careful about setting `restartable` to `TRUE`. You can query failed jobs by querying the `*_SCHEDULER_JOB_RUN_DETAILS` views.

Broken Jobs

A broken job is one that has exceeded a certain number of failures. This number is set in `max_failures`, and can be altered. In the case of a broken job, the entire job is broken, and it will not be run until it has been fixed. For debugging and testing, you can use the `RUN_JOB` procedure.

You can query broken jobs by querying the `*_SCHEDULER_JOBS` and `*_SCHEDULER_JOB_LOG` views.

Disabled Jobs

A job can become disabled for the following reasons:

- The job was manually disabled
- The job class it belongs to was dropped
- The program or schedule that it points to was dropped
- A window or window group is its schedule and it is dropped

Completed Jobs

A job will be completed if `end_date` or `max_runs` is reached.

How to Change Job Priorities

You can change job priorities by using the `SET_ATTRIBUTE` procedure. Job priorities must be in the range 1-5 with 1 being the highest priority. For example, the following statement changes the job priority for `my_job1` to a setting of 1:

```
BEGIN
DBMS_SCHEDULER.SET_ATTRIBUTE (
   name           =>   'my_emp_job1',
   attribute      =>   'job_priority',
   value          =>   1);
END;
/
```

You can verify that the attribute was changed by issuing the following statement:

```
SELECT JOB_NAME, JOB_PRIORITY FROM DBA_SCHEDULER_JOBS;

JOB_NAME                       JOB_PRIORITY
------------------------------ ------------
MY_EMP_JOB                                3
MY_EMP_JOB1                               1
MY_NEW_JOB1                               3
MY_NEW_JOB2                               3
MY_NEW_JOB3                               3
```

See Also: *PL/SQL Packages and Types Reference* for detailed information about the `SET_ATTRIBUTE` procedure

How the Scheduler Guarantees Availability

You do not need to perform any special operations for the Scheduler in the event of a system or slave process failure.

How to Handle Scheduler Security

You should grant the CREATE JOB system privilege to regular users who need to be able to use the Scheduler to schedule and run jobs. You should grant MANAGE SCHEDULER to any database administrator who needs to be able to manage system resources. Granting any other Scheduler system privilege or role should not be done without great caution. In particular, the CREATE ANY JOB system privilege and the SCHEDULER_ADMIN role, which includes it, are very powerful because they allow execution of code as any user. They should only be granted to very powerful roles or users.

A particularly important issue from a security point of view is handling external jobs. See "Running External Jobs" on page 27-8 for further information. Security for the Scheduler has no other special requirements. See *Oracle Database Security Guide* for details regarding security.

How to Manage the Scheduler in a RAC Environment

There are no special requirements for using the Scheduler in a RAC environment.

Import/Export and the Scheduler

You must use the Data Pump utilities (impdp and expdp) to export Scheduler objects. You cannot use the earlier import/export utilities with the Scheduler. Also, Scheduler objects cannot be exported while the database is in read-only mode.

An export generates the DDL that was used to create the Scheduler objects. All attributes are exported. When an import is done, all the database objects are recreated in the new database. All schedules are stored with their time zones, which are maintained in the new database. For example, schedule "Monday at 1 PM PST in a database in San Francisco" would be the same if it was exported and imported to a database in Germany.

> **See Also:** *Oracle Database Utilities* for details regarding import and export

Examples of Using the Scheduler

This section discusses the following topics:

- Examples of Creating Jobs
- Examples of Creating Job Classes
- Examples of Creating Programs
- Examples of Creating Windows
- Examples of Setting Attributes

Examples of Creating Jobs

This section contains several examples of creating jobs.

Example 28–1 Creating a Job

The following statement creates a job called `my_job1` in the `oe` schema:

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB (
   job_name             => 'oe.my_job1',
   job_type             => 'PLSQL_BLOCK',
   job_action           => 'DBMS_STATS.GATHER_TABLE_STATS(''oe'',''sales'');',
   start_date           => '15-JUL-03 1.00.00AM US/Pacific',
   repeat_interval      => 'FREQ=DAILY',
   end_date             => '15-SEP-03 1.00.00AM US/Pacific',
   enabled              =>  TRUE,
   comments             => 'Gather table statistics');
END;
/
```

This job gathers table statistics on the `sales` table. It will run for the first time on July 15th and then once a day until September 15. To verify that the job was created, issue the following statement:

```
SELECT JOB_NAME FROM DBA_SCHEDULER_JOBS WHERE JOB_NAME = 'MY_JOB1';

JOB_NAME
------------------------------
MY_JOB1
```

Example 28–2 Creating a Job

The following statement creates a job called my_job2 in the SYSTEM schema:

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB (
   job_name           =>  'system.my_job1',
   job_type           =>  'PLSQL_BLOCK',
   job_action         =>  'INSERT INTO sales VALUES( 7987, ''SALLY'',
      ''ANALYST'', NULL, NULL, NULL, NULL, NULL);',
   start_date         =>  '28-APR-03 07.00.00.000000 AM Europe/Warsaw',
   repeat_interval    =>  'FREQ=HOURLY;INTERVAL=2', /* every two hours */
   end_date           =>  '20-NOV-04 07.00.00.000000 AM Europe/Warsaw',
   comments           =>  'My new job');
END;
/
```

> **See Also:** *PL/SQL Packages and Types Reference* for detailed information about the CREATE_JOB procedure and "Creating Jobs" on page 27-3 for further information

Examples of Creating Job Classes

This section contains several examples of creating job classes.

Example 28–3 Creating a Job Class

The following statement creates a job class:

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB_CLASS (
   job_class_name             =>  'my_class1',
   service                    =>  'my_service1',
   comments                   =>  'This is my first job class');
END;
/
```

This creates my_class1 in SYS. It uses a service called my_service1. To verify that the job class was created, issue the following statement:

```
SELECT JOB_CLASS_NAME FROM DBA_SCHEDULER_JOB_CLASSES
WHERE JOB_CLASS_NAME = 'MY_CLASS1';

JOB_CLASS_NAME
------------------------------
MY_CLASS1
```

Example 28–4 Creating a Job Class

The following statement creates a job class:

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB_CLASS (
   job_class_name             =>  'finance_jobs',
   resource_consumer_group    =>  'finance_group',
   comments                   =>  'My financial class');
END;
/
```

This creates `finance_jobs` in SYS. It uses a resource consumer group called `finance_group`.

> **See Also:** *PL/SQL Packages and Types Reference* for detailed information about the `CREATE_JOB_CLASS` procedure and "Creating Job Classes" on page 27-24 for further information

Examples of Creating Programs

This section contains several examples of creating programs.

Example 28–5 Creating a Program

The following statement creates a program in the `oe` schema:

```
BEGIN
DBMS_SCHEDULER.CREATE_PROGRAM (
   program_name          => 'oe.my_program1',
   program_type          => 'PLSQL_BLOCK',
   program_action        => 'DBMS_STATS.GATHER_TABLE_STATS(''oe'',''sales'');',
   number_of_arguments   => 0,
   enabled               => TRUE,
   comments              => 'My comments here');
END;
/
```

This creates `my_program1`, which uses PL/SQL to gather table statistics on the `sales` table. To verify that the program was created, issue the following statement:

```
SELECT PROGRAM_NAME FROM DBA_SCHEDULER_PROGRAMS
WHERE PROGRAM_NAME = 'MY_PROGRAM1';

PROGRAM_NAME
------------------------
```

```
MY_PROGRAM1
```

Example 28–6 Creating a Program

The following statement creates a program in the oe schema:

```
BEGIN
DBMS_SCHEDULER.CREATE_PROGRAM (
   program_name           => 'oe.my_saved_program1',
   program_action         => '/usr/local/bin/date',
   program_type           => 'EXECUTABLE',
   comments               => 'My comments here');
END;
/
```

This creates my_saved_program1, which uses an executable.

> **See Also:** *PL/SQL Packages and Types Reference* for detailed information about the CREATE_PROGRAM procedure and "Creating Programs" on page 27-13 for further information

Examples of Creating Windows

This section contains several examples of creating windows.

Example 28–7 Creating a Window

The following statement creates a window called my_window1 in SYS:

```
BEGIN
DBMS_SCHEDULER.CREATE_WINDOW (
   window_name          =>  'my_window1',
   resource_plan        =>  'my_res_plan1',
   start_date           =>  '15-JUL-03 1.00.00AM US/Pacific',
   repeat_interval      =>  'FREQ=DAILY',
   end_date             =>  '15-SEP-03 1.00.00AM US/Pacific',
   duration             =>  interval '80' MINUTE,
   comments             =>  'This is my first window');
END;
/
```

This window will open once a day at 1AM for 80 minutes every day from May 15th to October 15th. To verify that the window was created, issue the following statement:

```
SELECT WINDOW_NAME FROM DBA_SCHEDULER_WINDOWS WHERE WINDOW_NAME = 'MY_WINDOW1';
```

```
WINDOW_NAME
------------------------------
MY_WINDOW1
```

Example 28–8 Creating a Window

The following statement creates a window called my_window2 in SYS:

```
BEGIN
DBMS_SCHEDULER.CREATE_WINDOW (
   window_name       => 'my_window2',
   schedule_name     => 'my_stats_schedule',
   resource_plan     => 'my_resourceplan1',
   duration          => interval '60' minute,
   comments          => 'My window');
END;
/
```

> **See Also:** *PL/SQL Packages and Types Reference* for detailed information about the CREATE_WINDOW procedure and "Creating Windows" on page 27-27 for further information

Example of Creating Window Groups

The following statement creates a window group called my_window_group1:

```
BEGIN
DBMS_SCHEDULER.CREATE_WINDOW_GROUP ('my_windowgroup1');
END;
/
```

Then, you could add three windows (my_window1, my_window2, and my_window3) to my_window_group1 by issuing the following statements:

```
BEGIN
DBMS_SCHEDULER.ADD_WINDOW_GROUP_MEMBER (
   group_name   =>  'my_window_group1',
   window_list  =>  'my_window1, my_window2');

DBMS_SCHEDULER.ADD_WINDOW_GROUP_MEMBER (
   group_name   =>  'my_window_group1',
   window_list  =>  'my_window3');
END;
/
```

To verify that the window group was created and the windows added to it, issue the following statement:

```
SELECT * FROM DBA_SCHEDULER_WINDOW_GROUPS;

WINDOW_GROUP_NAME    ENABLED   NUMBER_OF_WINDOWS   COMMENTS
-----------------    -------   -----------------   --------------
MY_WINDOW_GROUP1     TRUE                      3   This is my first window group
```

> **See Also:** *PL/SQL Packages and Types Reference* for detailed information about the CREATE_WINDOW_GROUP and ADD_WINDOW_GROUP_MEMBER procedures and "Creating Window Groups" on page 27-38 for further information

Examples of Setting Attributes

This section contains several examples of setting attributes.

Example 28–9 Setting the Frequency Attribute

The following example resets the frequency my_emp_job1 will run to daily:

```
BEGIN
DBMS_SCHEDULER.SET_ATTRIBUTE (
   name           =>   'my_emp_job1',
   attribute      =>   'repeat_interval',
   value          =>   'FREQ=DAILY');
END;
/
```

To verify the change, issue the following statement:

```
SELECT JOB_NAME, REPEAT_INTERVAL FROM DBA_SCHEDULER_JOBS
WHERE JOB_NAME =  'MY_EMP_JOB1';

JOB_NAME             REPEAT_INTERVAL
----------------     ---------------
MY_EMP_JOB1          FREQ=DAILY
```

Example 28–10 Setting the Comments Attribute

The following example resets the comments for my_saved_program1:

```
BEGIN
DBMS_SCHEDULER.SET_ATTRIBUTE (
   name           =>   'my_saved_program1',
```

```
   attribute      =>   'comments',
   value          =>   'For nightly table stats');
END;
/
```

To verify the change, issue the following statement:

```
SELECT PROGRAM_NAME, COMMENTS FROM DBA_SCHEDULER_PROGRAMS;

PROGRAM_NAME        COMMENTS
------------        ----------------------
MY_PROGRAM1         My comments here
MY_SAVED_PROGRAM1   For nightly table stats
```

Example 28–11 Setting the Duration Attribute

The following example resets the duration of `my_window3` to 90 minutes:

```
BEGIN
DBMS_SCHEDULER.SET_ATTRIBUTE (
   name           =>   'my_window3',
   attribute      =>   'duration',
   value          =>   interval '90' minute);
END;
/
```

To verify the change, issue the following statement:

```
SELECT WINDOW_NAME, DURATION FROM DBA_SCHEDULER_WINDOWS
WHERE WINDOW_NAME = 'MY_WINDOW3';

WINDOW_NAME        DURATION
-----------        ---------------
MY_WINDOW3         +000 00:90:00
```

> **See Also:** *PL/SQL Packages and Types Reference* for detailed information about the `SET_SCHEDULER_ATTRIBUTE` procedure and "Task 2E: Setting Scheduler Attributes" on page 28-6

Part VII

Distributed Database Management

Part VI discusses the management of a distributed database environment. It contains the following sections:

29

Distributed Database Concepts

This chapter describes the basic concepts and terminology of Oracle Database distributed database architecture. It contains the following topics:

- Distributed Database Architecture
- Database Links
- Distributed Database Administration
- Transaction Processing in a Distributed System
- Distributed Database Application Development
- Character Set Support for Distributed Environments

Distributed Database Architecture

A **distributed database system** allows applications to access data from local and remote databases. In a **homogenous distributed database system**, each database is an Oracle Database. In a **heterogeneous distributed database system**, at least one of the databases is not an Oracle Database. Distributed databases use a **client/server** architecture to process information requests.

This section contains the following topics:

- Homogenous Distributed Database Systems
- Heterogeneous Distributed Database Systems
- Client/Server Database Architecture

Homogenous Distributed Database Systems

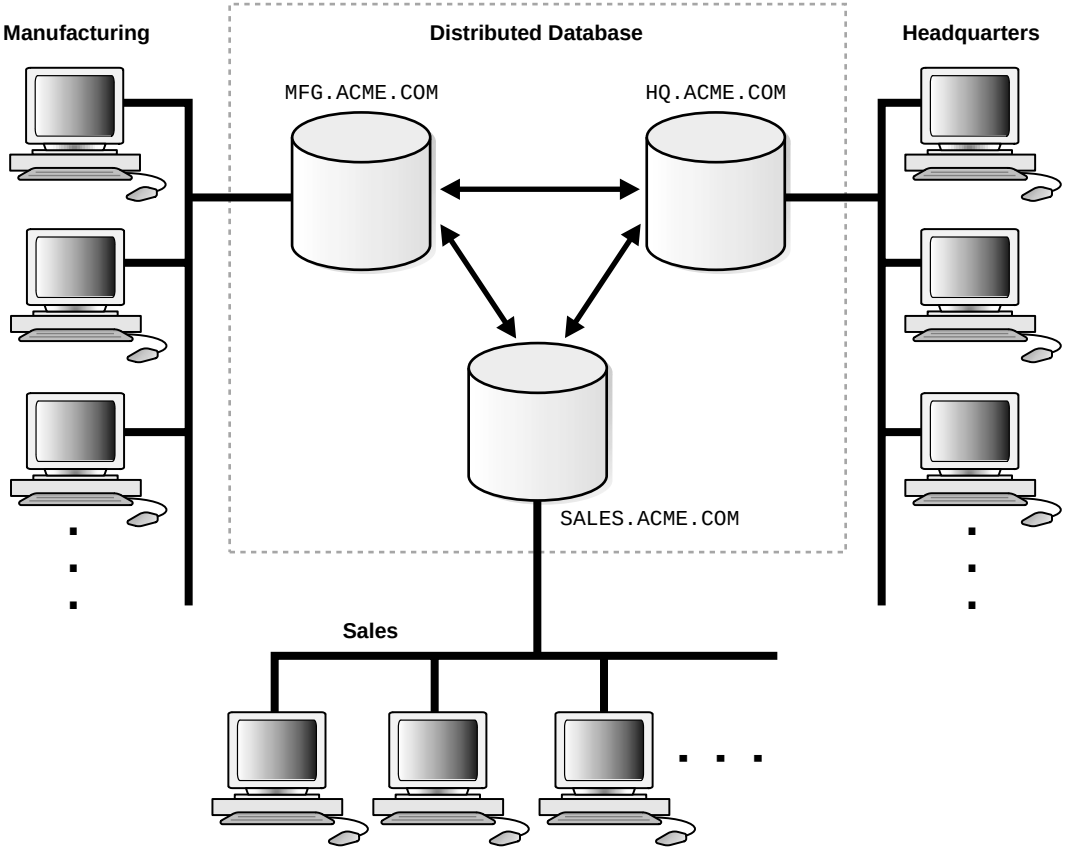
A homogenous distributed database system is a network of two or more Oracle Databases that reside on one or more machines. Figure 29–1 illustrates a distributed system that connects three databases: `hq`, `mfg`, and `sales`. An application can simultaneously access or modify the data in several databases in a single distributed environment. For example, a single query from a Manufacturing client on local database `mfg` can retrieve joined data from the `products` table on the local database and the `dept` table on the remote `hq` database.

For a client application, the location and platform of the databases are transparent. You can also create **synonyms** for remote objects in the distributed system so that users can access them with the same syntax as local objects. For example, if you are connected to database `mfg` but want to access data on database `hq`, creating a synonym on `mfg` for the remote `dept` table enables you to issue this query:

```
SELECT * FROM dept;
```

In this way, a distributed system gives the appearance of native data access. Users on `mfg` do not have to know that the data they access resides on remote databases.

Figure 29–1 Homogeneous Distributed Database



An Oracle Database distributed database system can incorporate Oracle Databases of different versions. All supported releases of Oracle Database can participate in a distributed database system. Nevertheless, the applications that work with the distributed database must understand the functionality that is available at each node in the system. A distributed database application cannot expect an Oracle7 database to understand the SQL extensions that are only available with Oracle Database.

Distributed Databases Versus Distributed Processing

The terms **distributed database** and **distributed processing** are closely related, yet have distinct meanings. There definitions are as follows:

- Distributed database

  A set of databases in a distributed system that can appear to applications as a single data source.

- Distributed processing

  The operations that occurs when an application distributes its tasks among different computers in a network. For example, a database application typically distributes front-end presentation tasks to client computers and allows a back-end database server to manage shared access to a database. Consequently, a distributed database application processing system is more commonly referred to as a client/server database application system.

Distributed database systems employ a distributed processing architecture. For example, an Oracle Database server acts as a client when it requests data that another Oracle Database server manages.

Distributed Databases Versus Replicated Databases

The terms distributed database system and **database replication** are related, yet distinct. In a **pure** (that is, not replicated) distributed database, the system manages a single copy of all data and supporting database objects. Typically, distributed database applications use distributed transactions to access both local and remote data and modify the global database in real-time.

Note: This book discusses only pure distributed databases.

The term **replication** refers to the operation of copying and maintaining database objects in multiple databases belonging to a distributed system. While replication relies on distributed database technology, database replication offers applications benefits that are not possible within a pure distributed database environment.

Most commonly, replication is used to improve local database performance and protect the availability of applications because alternate data access options exist. For example, an application may normally access a local database rather than a remote server to minimize network traffic and achieve maximum performance. Furthermore, the application can continue to function if the local server experiences a failure, but other servers with replicated data remain accessible.

See Also:

- *Oracle Database Advanced Replication* for more information about Oracle Database replication features
- *Oracle Streams Concepts and Administration* for information about Oracle Streams, another method of sharing information between databases

Heterogeneous Distributed Database Systems

In a heterogeneous distributed database system, at least one of the databases is a non-Oracle Database system. To the application, the heterogeneous distributed database system appears as a single, local, Oracle Database. The local Oracle Database server hides the distribution and heterogeneity of the data.

The Oracle Database server accesses the non-Oracle Database system using Oracle Heterogeneous Services in conjunction with an **agent**. If you access the non-Oracle Database data store using an Oracle Transparent Gateway, then the agent is a system-specific application. For example, if you include a Sybase database in an Oracle Database distributed system, then you need to obtain a Sybase-specific transparent gateway so that the Oracle Database in the system can communicate with it.

Alternatively, you can use **generic connectivity** to access non-Oracle Database data stores so long as the non-Oracle Database system supports the ODBC or OLE DB protocols.

Note: Other than the introductory material presented in this chapter, this book does not discuss Oracle Heterogeneous Services. See *Oracle Database Heterogeneous Connectivity Administrator's Guide* for more detailed information about Heterogeneous Services.

Heterogeneous Services

Heterogeneous Services (HS) is an integrated component within the Oracle Database server and the enabling technology for the current suite of Oracle Transparent Gateway products. HS provides the common architecture and administration mechanisms for Oracle Database gateway products and other heterogeneous access facilities. Also, it provides upwardly compatible functionality for users of most of the earlier Oracle Transparent Gateway releases.

Transparent Gateway Agents

For each non-Oracle Database system that you access, Heterogeneous Services can use a transparent gateway agent to interface with the specified non-Oracle Database system. The agent is specific to the non-Oracle Database system, so each type of system requires a different agent.

The transparent gateway agent facilitates communication between Oracle Database and non-Oracle Database systems and uses the Heterogeneous Services component in the Oracle Database server. The agent executes SQL and transactional requests at the non-Oracle Database system on behalf of the Oracle Database server.

> **See Also:** Your Oracle-supplied gateway-specific documentation for information about transparent gateways

Generic Connectivity

Generic connectivity enables you to connect to non-Oracle Database data stores by using either a Heterogeneous Services ODBC agent or a Heterogeneous Services OLE DB agent. Both are included with your Oracle product as a standard feature. Any data source compatible with the ODBC or OLE DB standards can be accessed using a generic connectivity agent.

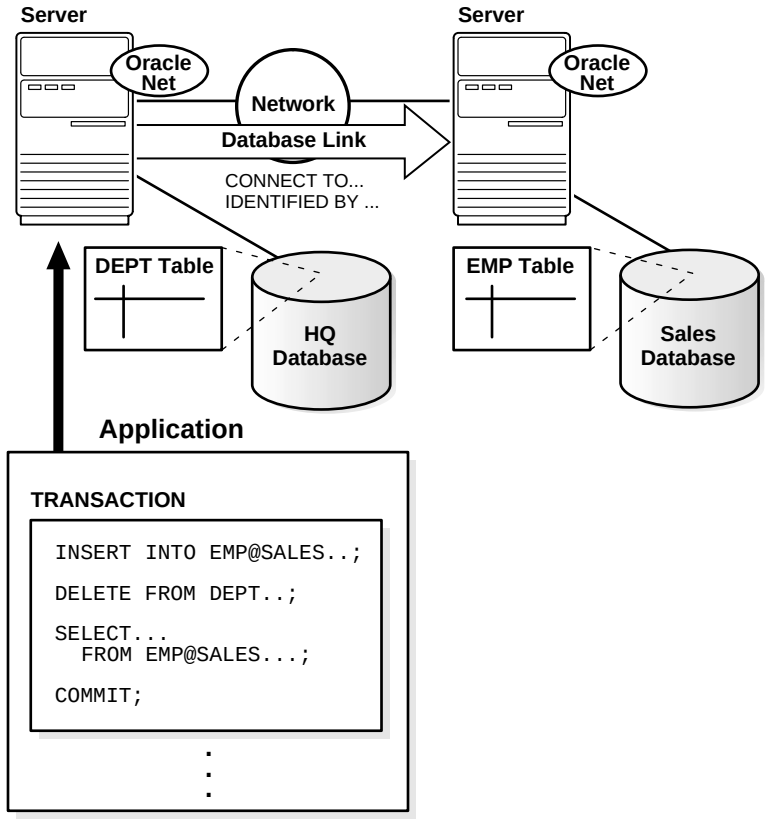
The advantage to generic connectivity is that it may not be required for you to purchase and configure a separate system-specific agent. You use an ODBC or OLE DB driver that can interface with the agent. However, some data access features are only available with transparent gateway agents.

Client/Server Database Architecture

A database server is the Oracle software managing a database, and a client is an application that requests information from a server. Each computer in a network is a node that can host one or more databases. Each node in a distributed database system can act as a client, a server, or both, depending on the situation.

In Figure 29–2, the host for the `hq` database is acting as a database server when a statement is issued against its local data (for example, the second statement in each transaction issues a statement against the local `dept` table), but is acting as a client when it issues a statement against remote data (for example, the first statement in each transaction is issued against the remote table `emp` in the `sales` database).

Figure 29–2 An Oracle Database Distributed Database System



A client can connect **directly** or **indirectly** to a database server. A direct connection occurs when a client connects to a server and accesses information from a database contained on that server. For example, if you connect to the `hq` database and access the `dept` table on this database as in Figure 29–2, you can issue the following:

```
SELECT * FROM dept;
```

This query is direct because you are not accessing an object on a remote database.

In contrast, an indirect connection occurs when a client connects to a server and then accesses information contained in a database on a different server. For example, if you connect to the `hq` database but access the `emp` table on the remote `sales` database as in Figure 29–2, you can issue the following:

```
SELECT * FROM emp@sales;
```

This query is indirect because the object you are accessing is not on the database to which you are directly connected.

Database Links

The central concept in distributed database systems is a **database link**. A database link is a connection between two physical database servers that allows a client to access them as one logical database.

This section contains the following topics:

What Are Database Links?

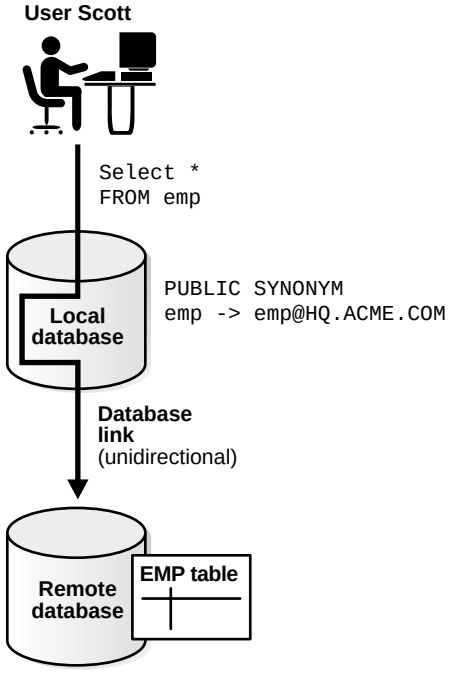
A database link is a pointer that defines a one-way communication path from an Oracle Database server to another database server. The link pointer is actually defined as an entry in a data dictionary table. To access the link, you must be connected to the local database that contains the data dictionary entry.

A database link connection is one-way in the sense that a client connected to local database A can use a link stored in database A to access information in remote database B, but users connected to database B cannot use the same link to access data in database A. If local users on database B want to access data on database A, then they must define a link that is stored in the data dictionary of database B.

A database link connection allows local users to access data on a remote database. For this connection to occur, each database in the distributed system must have a unique **global database name** in the network domain. The global database name uniquely identifies a database server in a distributed system.

Figure 29–3 shows an example of user `scott` accessing the `emp` table on the remote database with the global name `hq.acme.com`:

Figure 29–3 Database Link



Database links are either private or public. If they are private, then only the user who created the link has access; if they are public, then all database users have access.

One principal difference among database links is the way that connections to a remote database occur. Users access a remote database through the following types of links:

Type of Link	Description
Connected user link	Users connect as themselves, which means that they must have an account on the remote database with the same username as their account on the local database.

Type of Link	Description
Fixed user link	Users connect using the username and password referenced in the link. For example, if Jane uses a fixed user link that connects to the `hq` database with the username and password `scott/tiger`, then she connects as `scott`, Jane has all the privileges in `hq` granted to `scott` directly, and all the default roles that `scott` has been granted in the `hq` database.
Current user link	A user connects as a global user. A local user can connect as a global user in the context of a stored procedure, without storing the global user's password in a link definition. For example, Jane can access a procedure that Scott wrote, accessing Scott's account and Scott's schema on the `hq` database. Current user links are an aspect of Oracle Advanced Security.

Create database links using the `CREATE DATABASE LINK` statement. After a link is created, you can use it to specify schema objects in SQL statements.

See Also:

- *Oracle Database SQL Reference* for syntax of the `CREATE DATABASE` statement
- *Oracle Advanced Security Administrator's Guide* for information about Oracle Advanced Security

What Are Shared Database Links?

A shared database link is a link between a local server process and the remote database. The link is shared because multiple client processes can use the same link simultaneously.

When a local database is connected to a remote database through a database link, either database can run in dedicated or shared server mode. The following table illustrates the possibilities:

Local Database Mode	Remote Database Mode
Dedicated	Dedicated
Dedicated	Shared server
Shared server	Dedicated
Shared server	Shared server

A shared database link can exist in any of these four configurations. Shared links differ from standard database links in the following ways:

- Different users accessing the same schema object through a database link can share a network connection.
- When a user needs to establish a connection to a remote server from a particular server process, the process can reuse connections already established to the remote server. The reuse of the connection can occur if the connection was established on the same server process with the same database link, possibly in a different session. In a nonshared database link, a connection is not shared across multiple sessions.
- When you use a shared database link in a shared server configuration, a network connection is established directly out of the shared server process in the local server. For a nonshared database link on a local shared server, this connection would have been established through the local dispatcher, requiring context switches for the local dispatcher, and requiring data to go through the dispatcher.

> **See Also:** *Oracle Net Services Administrator's Guide* for information about shared server

Why Use Database Links?

The great advantage of database links is that they allow users to access another user's objects in a remote database so that they are bounded by the privilege set of the object owner. In other words, a local user can access a link to a remote database without having to be a user on the remote database.

For example, assume that employees submit expense reports to Accounts Payable (A/P), and further suppose that a user using an A/P application needs to retrieve information about employees from the `hq` database. The A/P users should be able to connect to the `hq` database and execute a stored procedure in the remote `hq` database that retrieves the desired information. The A/P users should not need to be `hq` database users to do their jobs; they should only be able to access `hq` information in a controlled way as limited by the procedure.

Database links allow you to grant limited access on remote databases to local users. By using current user links, you can create centrally managed global users whose password information is hidden from both administrators *and* nonadministrative users. For example, A/P users can access the `hq` database as `scott`, but unlike fixed user links, `scott`'s credentials are not stored where database users can see them.

By using fixed user links, you can create nonglobal users whose password information is stored in unencrypted form in the LINK$ data dictionary table. Fixed user links are easy to create and require low overhead because there are no SSL or directory requirements, but a security risk results from the storage of password information in the data dictionary.

See Also:

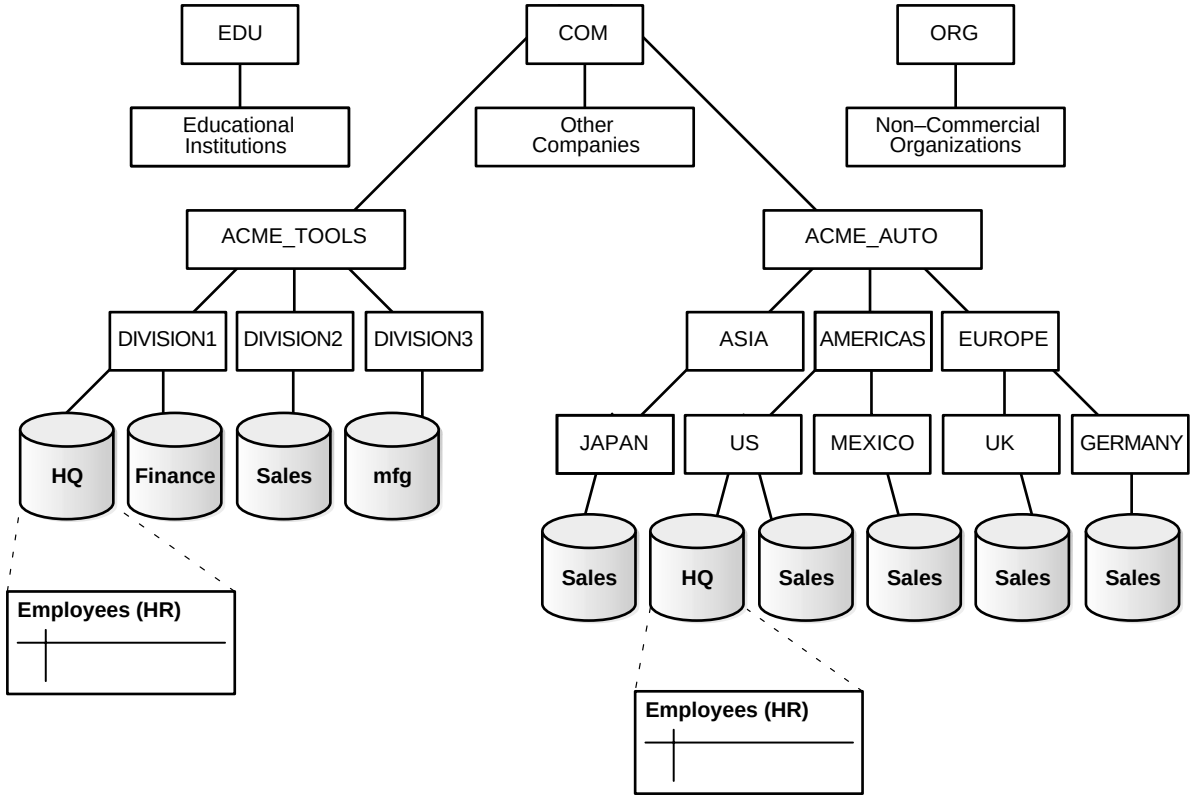
- "Users of Database Links" on page 29-16 for an explanation of database link users
- "Viewing Information About Database Links" for an explanation of how to hide passwords from nonadministrative users

Global Database Names in Database Links

To understand how a database link works, you must first understand what a global database name is. Each database in a distributed database is uniquely identified by its global database name. The database forms a global database name by prefixing the database network domain, specified by the DB_DOMAIN initialization parameter at database creation, with the individual database name, specified by the DB_NAME initialization parameter.

For example, Figure 29–4 illustrates a representative hierarchical arrangement of databases throughout a network.

Figure 29–4 Hierarchical Arrangement of Networked Databases



The name of a database is formed by starting at the leaf of the tree and following a path to the root. For example, the `mfg` database is in `division3` of the `acme_tools` branch of the `com` domain. The global database name for `mfg` is created by concatenating the nodes in the tree as follows:

- `mfg.division3.acme_tools.com`

While several databases can share an individual name, each database must have a unique global database name. For example, the network domains `us.americas.acme_auto.com` and `uk.europe.acme_auto.com` each contain a `sales` database. The global database naming system distinguishes the `sales` database in the `americas` division from the `sales` database in the `europe` division as follows:

- `sales.us.americas.acme_auto.com`

- `sales.uk.europe.acme_auto.com`

See Also: "Managing Global Names in a Distributed System" on page 30-1 to learn how to specify and change global database names

Names for Database Links

Typically, a database link has the same name as the global database name of the remote database that it references. For example, if the global database name of a database is `sales.us.oracle.com`, then the database link is also called `sales.us.oracle.com`.

When you set the initialization parameter `GLOBAL_NAMES` to `TRUE`, the database ensures that the name of the database link is the same as the global database name of the remote database. For example, if the global database name for `hq` is `hq.acme.com`, and `GLOBAL_NAMES` is `TRUE`, then the link name must be called `hq.acme.com`. Note that the database checks the domain part of the global database name as stored in the data dictionary, *not* the `DB_DOMAIN` setting in the initialization parameter file (see "Changing the Domain in a Global Database Name" on page 30-4).

If you set the initialization parameter `GLOBAL_NAMES` to `FALSE`, then you are not required to use global naming. You can then name the database link whatever you want. For example, you can name a database link to `hq.acme.com` as `foo`.

Note: Oracle recommends that you use global naming because many useful features, including Replication, require global naming.

After you have enabled global naming, database links are essentially transparent to users of a distributed database because the name of a database link is the same as the global name of the database to which the link points. For example, the following statement creates a database link in the local database to remote database `sales`:

```
CREATE PUBLIC DATABASE LINK sales.division3.acme.com USING 'sales1';
```

See Also: *Oracle Database Reference* for more information about specifying the initialization parameter `GLOBAL_NAMES`

Types of Database Links

Oracle Database lets you create **private**, **public**, and **global** database links. These basic link types differ according to which users are allowed access to the remote database:

Type	Owner	Description
Private	User who created the link. View ownership data through: ■ DBA_DB_LINKS ■ ALL_DB_LINKS ■ USER_DB_LINKS	Creates link in a specific schema of the local database. Only the owner of a private database link or PL/SQL subprograms in the schema can use this link to access database objects in the corresponding remote database.
Public	User called PUBLIC. View ownership data through views shown for private database links.	Creates a database-wide link. All users and PL/SQL subprograms in the database can use the link to access database objects in the corresponding remote database.
Global	User called PUBLIC. View ownership data through views shown for private database links.	Creates a network-wide link. When an Oracle network uses a directory server, the directory server automatically create and manages global database links (as net service names) for every Oracle Database in the network. Users and PL/SQL subprograms in any database can use a global link to access objects in the corresponding remote database. **Note**: In earlier releases of Oracle Database, a global database link referred to a database link that was registered with an Oracle Names server. The use of an Oracle Names server has been deprecated. In this document, global database links refer to the use of net service names from the directory server.

Determining the type of database links to employ in a distributed database depends on the specific requirements of the applications using the system. Consider these features when making your choice:

Type of Link	Features
Private database link	This link is more secure than a public or global link, because only the owner of the private link, or subprograms within the same schema, can use the link to access the remote database.

Type of Link	Features
Public database link	When many users require an access path to a remote Oracle Database, you can create a single public database link for all users in a database.
Global database link	When an Oracle network uses a directory server, an administrator can conveniently manage global database links for all databases in the system. Database link management is centralized and simple.

See Also:

- "Specifying Link Types" on page 30-9 to learn how to create different types of database links
- "Viewing Information About Database Links" on page 30-21 to learn how to access information about links

Users of Database Links

When creating the link, you determine which user should connect to the remote database to access the data. The following table explains the differences among the categories of users involved in database links:

User Type	Description	Sample Link Creation Syntax
Connected user	A local user accessing a database link in which no fixed username and password have been specified. If `SYSTEM` accesses a public link in a query, then the connected user is `SYSTEM`, and the database connects to the `SYSTEM` schema in the remote database. **Note:** A connected user does not have to be the user who created the link, but is any user who is accessing the link.	`CREATE PUBLIC DATABASE LINK hq USING 'hq';`
Current user	A global user in a `CURRENT_USER` database link. The global user must be authenticated by an X.509 certificate (an SSL-authenticated enterprise user) or a password (a password-authenticated enterprise user), and be a user on both databases involved in the link. Current user links are an aspect of the Oracle Advanced Security option. See *Oracle Advanced Security Administrator's Guide* for information about global security	`CREATE PUBLIC DATABASE LINK hq CONNECT TO CURRENT_USER using 'hq';`

User Type	Description	Sample Link Creation Syntax
Fixed user	A user whose username / password is part of the link definition. If a link includes a fixed user, then the fixed user's username and password are used to connect to the remote database.	`CREATE PUBLIC DATABASE LINK hq CONNECT TO jane IDENTIFIED BY doe USING 'hq';`

See Also: "Specifying Link Users" on page 30-11 to learn how to specify users where creating links

Connected User Database Links

Connected user links have no connect string associated with them. The advantage of a connected user link is that a user referencing the link connects to the remote database as the same user. Furthermore, because no connect string is associated with the link, no password is stored in clear text in the data dictionary.

Connected user links have some disadvantages. Because these links require users to have accounts and privileges on the remote databases to which they are attempting to connect, they require more privilege administration for administrators. Also, giving users more privileges than they need violates the fundamental security concept of least privilege: users should only be given the privileges they need to perform their jobs.

The ability to use a connected user database link depends on several factors, chief among them whether the user is authenticated by the database using a password, or externally authenticated by the operating system or a network authentication service. If the user is externally authenticated, then the ability to use a connected user link also depends on whether the remote database accepts remote authentication of users, which is set by the `REMOTE_OS_AUTHENT` initialization parameter.

The `REMOTE_OS_AUTHENT` parameter operates as follows:

REMOTE_OS_AUTHENT Value	Consequences
`TRUE` for the remote database	An externally-authenticated user can connect to the remote database using a connected user database link.

REMOTE_OS_AUTHENT Value	Consequences
FALSE for the remote database	An externally-authenticated user cannot connect to the remote database using a connected user database link unless a secure protocol or a network authentication service supported by the Oracle Advanced Security option is used.

Fixed User Database Links

A benefit of a fixed user link is that it connects a user in a primary database to a remote database with the security context of the user specified in the connect string. For example, local user `joe` can create a public database link in `joe`'s schema that specifies the fixed user `scott` with password `tiger`. If `jane` uses the fixed user link in a query, then `jane` is the user on the local database, but she connects to the remote database as `scott/tiger`.

Fixed user links have a username and password associated with the connect string. The username and password are stored in unencrypted form in the data dictionary in the `LINK$` table.

Caution: The fact that the username and password are stored in unencrypted form in the data dictionary creates a potential security weakness of fixed user database links.

If the `07_DICTIONARY_ACCESSIBILITY` initialization parameter is set to `TRUE`, a user with the `SELECT ANY TABLE` system privilege has access to the data dictionary, and thus the authentication associated with a fixed user is compromised.

The default for the `07_DICTIONARY_ACCESSIBILITY` initialization parameter is `FALSE`.

For an example of this security problem, assume that `jane` does not have privileges to use a private link that connects to the `hq` database as `scott/tiger`, but has `SELECT ANY TABLE` privilege on a database in which the `07_DICTIONARY_ACCESSIBILITY` initialization parameter is set to `TRUE`. She can select from `LINK$` and read that the connect string to `hq` is `scott/tiger`. If `jane` has an account on the host on which `hq` resides, then she can connect to the host and then connect to `hq` as `scott` using the password `tiger`. She will have all `scott`'s privileges if she connects locally and any audit records will be recorded as if she were `scott`.

See Also: *Oracle Database Security Guide* for more information about system privileges and the O7_DICTIONARY_ACCESSIBILITY initialization parameter

Current User Database Links

Current user database links make use of a global user. A global user must be authenticated by an X.509 certificate or a password, and be a user on both databases involved in the link.

The user invoking the CURRENT_USER link does not have to be a global user. For example, if jane is authenticated (not as a global user) by password to the Accounts Payable database, she can access a stored procedure to retrieve data from the hq database. The procedure uses a current user database link, which connects her to hq as global user scott. User scott is a global user and authenticated through a certificate over SSL, but jane is not.

Note that current user database links have these consequences:

- If the current user database link is *not* accessed from within a stored object, then the current user is the same as the connected user accessing the link. For example, if scott issues a SELECT statement through a current user link, then the current user is scott.
- When executing a stored object such as a procedure, view, or trigger that accesses a database link, the current user is the user that *owns* the stored object, and not the user that *calls* the object. For example, if jane calls procedure scott.p (created by scott), and a current user link appears *within* the called procedure, then scott is the current user of the link.
- If the stored object is an invoker-rights function, procedure, or package, then the invoker's authorization ID is used to connect as a remote user. For example, if user jane calls procedure scott.p (an invoker-rights procedure created by scott), and the link appears inside procedure scott.p, then jane is the current user.
- You cannot connect to a database as an enterprise user and then use a current user link in a stored procedure that exists in a shared, global schema. For example, if user jane accesses a stored procedure in the shared schema guest on database hq, she cannot use a current user link in this schema to log on to a remote database.

See Also:

- "Distributed Database Security" on page 29-25 for more information about security issues relating to database links
- *Oracle Advanced Security Administrator's Guide*
- *PL/SQL User's Guide and Reference* for more information about invoker-rights functions, procedures, or packages.

Creation of Database Links: Examples

Create database links using the `CREATE DATABASE LINK` statement. The table gives examples of SQL statements that create database links in a local database to the remote `sales.us.americas.acme_auto.com` database:

SQL Statement	Connects To Database	Connects As	Link Type
`CREATE DATABASE LINK sales.us.americas.acme_ auto.com USING 'sales_us';`	`sales` using net service name `sales_us`	Connected user	Private connected user
`CREATE DATABASE LINK foo CONNECT TO CURRENT_USER USING 'am_sls';`	`sales` using service name `am_sls`	Current global user	Private current user
`CREATE DATABASE LINK sales.us.americas.acme_ auto.com CONNECT TO scott IDENTIFIED BY tiger USING 'sales_us';`	`sales` using net service name `sales_us`	`scott` using password `tiger`	Private fixed user
`CREATE PUBLIC DATABASE LINK sales CONNECT TO scott IDENTIFIED BY tiger USING 'rev';`	`sales` using net service name `rev`	`scott` using password `tiger`	Public fixed user
`CREATE SHARED PUBLIC DATABASE LINK sales.us.americas.acme_ auto.com CONNECT TO scott IDENTIFIED BY tiger AUTHENTICATED BY anupam IDENTIFIED BY bhide USING 'sales';`	`sales` using net service name `sales`	`scott` using password `tiger`, authenticated as `anupam` using password `bhide`	Shared public fixed user

See Also:

- "Creating Database Links" on page 30-8 to learn how to create link
- *Oracle Database SQL Reference* for information about the CREATE DATABASE LINK statement syntax

Schema Objects and Database Links

After you have created a database link, you can execute SQL statements that access objects on the remote database. For example, to access remote object `emp` using database link `foo`, you can issue:

```
SELECT * FROM emp@foo;
```

You must also be authorized in the remote database to access specific remote objects.

Constructing properly formed object names using database links is an essential aspect of data manipulation in distributed systems.

Naming of Schema Objects Using Database Links

Oracle Database uses the global database name to name the schema objects globally using the following scheme:

schema.schema_object@global_database_name

where:

- *schema* is a collection of logical structures of data, or schema objects. A schema is owned by a database user and has the same name as that user. Each user owns a single schema.
- *schema_object* is a logical data structure like a table, index, view, synonym, procedure, package, or a database link.
- *global_database_name* is the name that uniquely identifies a remote database. This name must be the same as the concatenation of the remote database initialization parameters DB_NAME and DB_DOMAIN, unless the parameter GLOBAL_NAMES is set to FALSE, in which case any name is acceptable.

For example, using a database link to database `sales.division3.acme.com`, a user or application can reference remote data as follows:

```
SELECT * FROM scott.emp@sales.division3.acme.com;  # emp table in scott's schema
SELECT loc FROM scott.dept@sales.division3.acme.com;
```

If GLOBAL_NAMES is set to FALSE, then you can use any name for the link to sales.division3.acme.com. For example, you can call the link foo. Then, you can access the remote database as follows:

```
SELECT name FROM scott.emp@foo;  # link name different from global name
```

Authorization for Accessing Remote Schema Objects

To access a remote schema object, you must be granted access to the remote object in the remote database. Further, to perform any updates, inserts, or deletes on the remote object, you must be granted the SELECT privilege on the object, along with the UPDATE, INSERT, or DELETE privilege. Unlike when accessing a local object, the SELECT privilege is necessary for accessing a remote object because the database has no remote describe capability. The database must do a SELECT * on the remote object in order to determine its structure.

Synonyms for Schema Objects

Oracle Database lets you create synonyms so that you can hide the database link name from the user. A synonym allows access to a table on a remote database using the same syntax that you would use to access a table on a local database. For example, assume you issue the following query against a table in a remote database:

```
SELECT * FROM emp@hq.acme.com;
```

You can create the synonym emp for emp@hq.acme.com so that you can issue the following query instead to access the same data:

```
SELECT * FROM emp;
```

> **See Also:** "Using Synonyms to Create Location Transparency" on page 30-28 to learn how to create synonyms for objects specified using database links

Schema Object Name Resolution

To resolve application references to schema objects (a process called **name resolution**), the database forms object names hierarchically. For example, the database guarantees that each schema within a database has a unique name, and that within a schema each object has a unique name. As a result, a schema object name is always unique within the database. Furthermore, the database resolves application references to the local name of the object.

In a distributed database, a schema object such as a table is accessible to all applications in the system. The database extends the hierarchical naming model with global database names to effectively create **global object names** and resolve references to the schema objects in a distributed database system. For example, a query can reference a remote table by specifying its fully qualified name, including the database in which it resides.

For example, assume that you connect to the local database as user SYSTEM:

```
CONNECT SYSTEM/password@sales1
```

You then issue the following statements using database link hq.acme.com to access objects in the scott and jane schemas on remote database hq:

```
SELECT * FROM scott.emp@hq.acme.com;
INSERT INTO jane.accounts@hq.acme.com (acc_no, acc_name, balance)
  VALUES (5001, 'BOWER', 2000);
UPDATE jane.accounts@hq.acme.com
  SET balance = balance + 500;
DELETE FROM jane.accounts@hq.acme.com
  WHERE acc_name = 'BOWER';
```

Database Link Restrictions

You *cannot* perform the following operations using database links:

- Grant privileges on remote objects
- Execute DESCRIBE operations on some remote objects. The following remote objects, however, do support DESCRIBE operations:
  - Tables
  - Views
  - Procedures
  - Functions
- Analyze remote objects
- Define or enforce referential integrity
- Grant roles to users in a remote database
- Obtain nondefault roles on a remote database. For example, if jane connects to the local database and executes a stored procedure that uses a fixed user link

connecting as `scott`, `jane` receives `scott`'s default roles on the remote database. Jane cannot issue `SET ROLE` to obtain a nondefault role.

- Execute hash query joins that use shared server connections
- Use a current user link without authentication through SSL, password, or NT native authentication

Distributed Database Administration

The following sections explain some of the topics relating to database management in an Oracle Database distributed database system:

- Site Autonomy
- Distributed Database Security
- Auditing Database Links
- Administration Tools

See Also:

- Chapter 30, "Managing a Distributed Database" to learn how to administer homogenous systems
- *Oracle Database Heterogeneous Connectivity Administrator's Guide* to learn about heterogeneous services concepts

Site Autonomy

Site autonomy means that each server participating in a distributed database is administered independently from all other databases. Although several databases can work together, each database is a separate repository of data that is managed individually. Some of the benefits of site autonomy in an Oracle Database distributed database include:

- Nodes of the system can mirror the logical organization of companies or groups that need to maintain independence.
- Local administrators control corresponding local data. Therefore, each database administrator's domain of responsibility is smaller and more manageable.
- Independent failures are less likely to disrupt other nodes of the distributed database. No single database failure need halt all distributed operations or be a performance bottleneck.

- Administrators can recover from isolated system failures independently from other nodes in the system.
- A data dictionary exists for each local database. A global catalog is not necessary to access local data.
- Nodes can upgrade software independently.

Although Oracle Database permits you to manage each database in a distributed database system independently, you should not ignore the global requirements of the system. For example, you may need to:

- Create additional user accounts in each database to support the links that you create to facilitate server-to-server connections.
- Set additional initialization parameters such as `COMMIT_POINT_STRENGTH`, and `OPEN_LINKS`.

Distributed Database Security

The database supports all of the security features that are available with a nondistributed database environment for distributed database systems, including:

- Password authentication for users and roles
- Some types of external authentication for users and roles including:
  - Kerberos version 5 for connected user links
  - DCE for connected user links
- Login packet encryption for client-to-server and server-to-server connections

The following sections explain some additional topics to consider when configuring an Oracle Database distributed database system:

- Authentication Through Database Links
- Authentication Without Passwords
- Supporting User Accounts and Roles
- Centralized User and Privilege Management
- Data Encryption

> **See Also:** *Oracle Advanced Security Administrator's Guide* for more information about external authentication

Authentication Through Database Links

Database links are either private or public, **authenticated** or **nonauthenticated**. You create public links by specifying the PUBLIC keyword in the link creation statement. For example, you can issue:

```
CREATE PUBLIC DATABASE LINK foo USING 'sales';
```

You create authenticated links by specifying the CONNECT TO clause, AUTHENTICATED BY clause, or both clauses together in the database link creation statement. For example, you can issue:

```
CREATE DATABASE LINK sales CONNECT TO scott IDENTIFIED BY tiger USING 'sales';
CREATE SHARED PUBLIC DATABASE LINK sales CONNECT TO mick IDENTIFIED BY jagger
     AUTHENTICATED BY david IDENTIFIED BY bowie USING 'sales';
```

This table describes how users access the remote database through the link:

Link Type	Authenticated	Security Access
Private	No	When connecting to the remote database, the database uses security information (userid / password) taken from the local session. Hence, the link is a connected user database link. Passwords must be synchronized between the two databases.
Private	Yes	The userid / password is taken from the link definition rather than from the local session context. Hence, the link is a fixed user database link. This configuration allows passwords to be different on the two databases, but the local database link password must match the remote database password. The password is stored in clear text on the local system catalog, adding a security risk.
Public	No	Works the same as a private nonauthenticated link, except that all users can reference this pointer to the remote database.
Public	Yes	All users on the local database can access the remote database and all use the same userid / password to make the connection. Also, the password is stored in clear text in the local catalog, so you can see the password if you have sufficient privileges in the local database.

Authentication Without Passwords

When using a connected user or current user database link, you can use an external authentication source such as Kerberos to obtain **end-to-end security**. In end-to-end authentication, credentials are passed from server to server and can be authenticated by a database server belonging to the same domain. For example, if `jane` is authenticated externally on a local database, and wants to use a connected user link to connect as herself to a remote database, the local server passes the security ticket to the remote database.

Supporting User Accounts and Roles

In a distributed database system, you must carefully plan the user accounts and roles that are necessary to support applications using the system. Note that:

- The user accounts necessary to establish server-to-server connections must be available in all databases of the distributed database system.
- The roles necessary to make available application privileges to distributed database application users must be present in all databases of the distributed database system.

As you create the database links for the nodes in a distributed database system, determine which user accounts and roles each site needs to support server-to-server connections that use the links.

In a distributed environment, users typically require access to many network services. When you must configure separate authentications for each user to access each network service, security administration can become unwieldy, especially for large systems.

> **See Also:** "Creating Database Links" on page 30-8 for more information about the user accounts that must be available to support different types of database links in the system

Centralized User and Privilege Management

The database provides different ways for you to manage the users and privileges involved in a distributed system. For example, you have these options:

- Enterprise user management. You can create global users who are authenticated through SSL or by using passwords, then manage these users and their privileges in a directory through an independent enterprise directory service.
- Network authentication service. This common technique simplifies security management for distributed environments. You can use the Oracle Advanced

Security option to enhance Oracle Net and the security of an Oracle Database distributed database system. Windows NT native authentication is an example of a non-Oracle authentication solution.

> **See Also:** *Oracle Advanced Security Administrator's Guide* for more information about global user security

Schema-Dependent Global Users One option for centralizing user and privilege management is to create the following:

- A global user in a centralized directory
- A user in every database that the global user must connect to

For example, you can create a global user called `fred` with the following SQL statement:

```
CREATE USER fred IDENTIFIED GLOBALLY AS 'CN=fred adams,O=Oracle,C=England';
```

This solution allows a single global user to be authenticated by a centralized directory.

The schema-dependent global user solution has the consequence that you must create a user called `fred` on every database that this user must access. Because most users need permission to access an application schema but do not need their own schemas, the creation of a separate account in each database for every global user creates significant overhead. Because of this problem, the database also supports schema-independent users, which are global users that an access a single, generic schema in every database.

Schema-Independent Global Users The database supports functionality that allows a global user to be centrally managed by an enterprise directory service. Users who are managed in the directory are called **enterprise users**. This directory contains information about:

- Which databases in a distributed system an enterprise user can access
- Which role on each database an enterprise user can use
- Which schema on each database an enterprise user can connect to

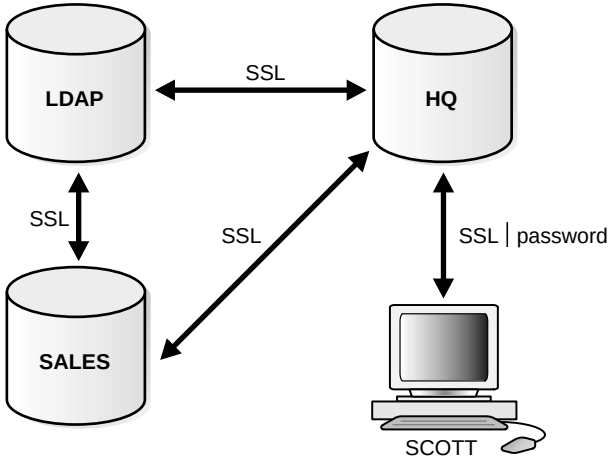
The administrator of each database is not required to create a global user account for each enterprise user on each database to which the enterprise user needs to connect. Instead, multiple enterprise users can connect to the same database schema, called a **shared schema**.

Note: You cannot access a current user database link in a shared schema.

For example, suppose jane, bill, and scott all use a human resources application. The hq application objects are all contained in the guest schema on the hq database. In this case, you can create a local global user account to be used as a shared schema. This global username, that is, shared schema name, is guest. jane, bill, and scott are all created as enterprise users in the directory service. They are also mapped to the guest schema in the directory, and can be assigned different authorizations in the hq application.

Figure 29–5 illustrates an example of global user security using the enterprise directory service:

Figure 29–5 Global User Security



Assume that the enterprise directory service contains the following information on enterprise users for hq and sales:

Database	Role	Schema	Enterprise Users
tb	clerk1	guest	bill scott

Database	Role	Schema	Enterprise Users
sales	clerk2	guest	jane scott

Also, assume that the local administrators for hq and sales have issued statements as follows:

Database	CREATE Statements
hq	CREATE USER guest IDENTIFIED GLOBALLY AS ''; CREATE ROLE clerk1 GRANT select ON emp; CREATE PUBLIC DATABASE LINK sales_link CONNECT AS CURRENT_USER USING 'sales';
sales	CREATE USER guest IDENTIFIED GLOBALLY AS ''; CREATE ROLE clerk2 GRANT select ON dept;

Assume that enterprise user scott requests a connection to local database hq in order to execute a distributed transaction involving sales. The following steps occur (not necessarily in this exact order):

1. Enterprise user scott is authenticated using SSL or a password.
2. User scott issues the following statement:

```
SELECT e.ename, d.loc
FROM emp e, dept@sales_link d
WHERE e.deptno=d.deptno;
```

3. Databases hq and sales mutually authenticate one another using SSL.
4. Database hq queries the enterprise directory service to determine whether enterprise user scott has access to hq, and discovers scott can access local schema guest using role clerk1.
5. Database sales queries the enterprise directory service to determine whether enterprise user scott has access to sales, and discovers scott can access local schema guest using role clerk2.
6. Enterprise user scott logs into sales to schema guest with role clerk2 and issues a SELECT to obtain the required information and transfer it to hq.
7. Database hq receives the requested data from sales and returns it to the client scott.

> **See Also:** *Oracle Advanced Security Administrator's Guide* for more information about enterprise user security

Data Encryption

The Oracle Advanced Security option also enables Oracle Net and related products to use network data encryption and checksumming so that data cannot be read or altered. It protects data from unauthorized viewing by using the RSA Data Security RC4 or the Data Encryption Standard (DES) encryption algorithm.

To ensure that data has not been modified, deleted, or replayed during transmission, the security services of the Oracle Advanced Security option can generate a cryptographically secure message digest and include it with each packet sent across the network.

> **See Also:** *Oracle Advanced Security Administrator's Guide* for more information about these and other features of the Oracle Advanced Security option

Auditing Database Links

You must always perform auditing operations locally. That is, if a user acts in a local database and accesses a remote database through a database link, the local actions are audited in the local database, and the remote actions are audited in the remote database, provided appropriate audit options are set in the respective databases.

The remote database cannot determine whether a successful connect request and subsequent SQL statements come from another server or from a locally connected client. For example, assume the following:

- Fixed user link `hq.acme.com` connects local user `jane` to the remote `hq` database as remote user `scott.`
- User `scott` is audited on the remote database.

Actions performed during the remote database session are audited as if `scott` were connected locally to `hq` and performing the same actions there. You must set audit options in the remote database to capture the actions of the username--in this case, `scott` on the `hq` database--embedded in the link if the desired effect is to audit what `jane` is doing in the remote database.

Note: You can audit the global username for global users.

You cannot set local auditing options on remote objects. Therefore, you cannot audit use of a database link, although access to remote objects can be audited on the remote database.

Administration Tools

The database administrator has several choices for tools to use when managing an Oracle Database distributed database system:

- Enterprise Manager
- Third-Party Administration Tools
- SNMP Support

Enterprise Manager

Enterprise Manager is the Oracle Database administration tool that provides a graphical user interface (GUI). Enterprise Manager provides administrative functionality for distributed databases through an easy-to-use interface. You can use Enterprise Manager to:

- Administer multiple databases. You can use Enterprise Manager to administer a single database or to simultaneously administer multiple databases.
- Centralize database administration tasks. You can administer both local and remote databases running on any Oracle Database platform in any location worldwide. In addition, these Oracle Database platforms can be connected by any network protocols supported by Oracle Net.
- Dynamically execute SQL, PL/SQL, and Enterprise Manager commands. You can use Enterprise Manager to enter, edit, and execute statements. Enterprise Manager also maintains a history of statements executed.

  Thus, you can reexecute statements without retyping them, a particularly useful feature if you need to execute lengthy statements repeatedly in a distributed database system.
- Manage security features such as global users, global roles, and the enterprise directory service.

Third-Party Administration Tools

Currently more than 60 companies produce more than 150 products that help manage Oracle Databases and networks, providing a truly open environment.

SNMP Support

Besides its network administration capabilities, Oracle **Simple Network Management Protocol** (*SNMP*) support allows an Oracle Database server to be located and queried by any SNMP-based network management system. SNMP is the accepted standard underlying many popular network management systems such as:

- HP OpenView
- Digital POLYCENTER Manager on NetView
- IBM NetView / 6000
- Novell NetWare Management System
- SunSoft SunNet Manager

> **See Also:** *Oracle SNMP Support Reference Guide* for more information about SNMP

Transaction Processing in a Distributed System

A transaction is a logical unit of work constituted by one or more SQL statements executed by a single user. A transaction begins with the user's first executable SQL statement and ends when it is committed or rolled back by that user.

A **remote transaction** contains only statements that access a single remote node. A **distributed transaction** contains statements that access more than one node.

The following sections define important concepts in transaction processing and explain how transactions access data in a distributed database:

- Remote SQL Statements
- Distributed SQL Statements
- Shared SQL for Remote and Distributed Statements
- Remote Transactions
- Distributed Transactions
- Two-Phase Commit Mechanism
- Database Link Name Resolution
- Schema Object Name Resolution

Remote SQL Statements

A **remote query** statement is a query that selects information from one or more remote tables, all of which reside at the same remote node. For example, the following query accesses data from the `dept` table in the `scott` schema of the remote `sales` database:

```
SELECT * FROM scott.dept@sales.us.americas.acme_auto.com;
```

A **remote update** statement is an update that modifies data in one or more tables, all of which are located at the same remote node. For example, the following query updates the `dept` table in the `scott` schema of the remote `sales` database:

```
UPDATE scott.dept@mktng.us.americas.acme_auto.com
  SET loc = 'NEW YORK'
  WHERE deptno = 10;
```

Note: A remote update can include a subquery that retrieves data from one or more remote nodes, but because the update happens at only a single remote node, the statement is classified as a remote update.

Distributed SQL Statements

A **distributed query** statement retrieves information from two or more nodes. For example, the following query accesses data from the local database as well as the remote `sales` database:

```
SELECT ename, dname
  FROM scott.emp e, scott.dept@sales.us.americas.acme_auto.com d
  WHERE e.deptno = d.deptno;
```

A **distributed update** statement modifies data on two or more nodes. A distributed update is possible using a PL/SQL subprogram unit such as a procedure or trigger that includes two or more remote updates that access data on different nodes. For example, the following PL/SQL program unit updates tables on the local database and the remote `sales` database:

```
BEGIN
  UPDATE scott.dept@sales.us.americas.acme_auto.com
    SET loc = 'NEW YORK'
    WHERE deptno = 10;
  UPDATE scott.emp
    SET deptno = 11
```

```
    WHERE deptno = 10;
END;
COMMIT;
```

The database sends statements in the program to the remote nodes, and their execution succeeds or fails as a unit.

Shared SQL for Remote and Distributed Statements

The mechanics of a remote or distributed statement using shared SQL are essentially the same as those of a local statement. The SQL text must match, and the referenced objects must match. If available, shared SQL areas can be used for the local and remote handling of any statement or decomposed query.

> **See Also:** *Oracle Database Concepts* for more information about shared SQL

Remote Transactions

A remote transaction contains one or more remote statements, all of which reference a single remote node. For example, the following transaction contains two statements, each of which accesses the remote `sales` database:

```
UPDATE scott.dept@sales.us.americas.acme_auto.com
  SET loc = 'NEW YORK'
  WHERE deptno = 10;
UPDATE scott.emp@sales.us.americas.acme_auto.com
  SET deptno = 11
  WHERE deptno = 10;
COMMIT;
```

Distributed Transactions

A distributed transaction is a transaction that includes one or more statements that, individually or as a group, update data on two or more distinct nodes of a distributed database. For example, this transaction updates the local database and the remote `sales` database:

```
UPDATE scott.dept@sales.us.americas.acme_auto.com
  SET loc = 'NEW YORK'
  WHERE deptno = 10;
UPDATE scott.emp
  SET deptno = 11
  WHERE deptno = 10;
```

```
COMMIT;
```

Note: If all statements of a transaction reference only a single remote node, the transaction is remote, not distributed.

Two-Phase Commit Mechanism

A database must guarantee that all statements in a transaction, distributed or nondistributed, either commit or roll back as a unit. The effects of an ongoing transaction should be invisible to all other transactions at all nodes; this transparency should be true for transactions that include any type of operation, including queries, updates, or remote procedure calls.

The general mechanisms of transaction control in a nondistributed database are discussed in the *Oracle Database Concepts*. In a distributed database, the database must coordinate transaction control with the same characteristics over a network and maintain data consistency, even if a network or system failure occurs.

The database **two-phase commit** mechanism guarantees that *all* database servers participating in a distributed transaction either all commit or all roll back the statements in the transaction. A two-phase commit mechanism also protects implicit DML operations performed by integrity constraints, remote procedure calls, and triggers.

See Also: Chapter 32, "Distributed Transactions Concepts" for more information about the Oracle Database two-phase commit mechanism

Database Link Name Resolution

A **global object name** is an object specified using a database link. The essential components of a global object name are:

- Object name
- Database name
- Domain

The following table shows the components of an explicitly specified global database object name:

Statement	Object	Database	Domain
SELECT * FROM joan.dept@sales.acme.com	dept	sales	acme.com
SELECT * FROM emp@mktg.us.acme.com	emp	mktg	us.acme.com

Whenever a SQL statement includes a reference to a global object name, the database searches for a database link with a name that matches the database name specified in the global object name. For example, if you issue the following statement:

```
SELECT * FROM scott.emp@orders.us.acme.com;
```

The database searches for a database link called `orders.us.acme.com`. The database performs this operation to determine the path to the specified remote database.

The database always searches for matching database links in the following order:

1. Private database links in the schema of the user who issued the SQL statement.
2. Public database links in the local database.
3. Global database links (only if a directory server is available).

Name Resolution When the Global Database Name Is Complete

Assume that you issue the following SQL statement, which specifies a complete global database name:

```
SELECT * FROM emp@prod1.us.oracle.com;
```

In this case, both the database name (`prod1`) and domain components (`us.oracle.com`) are specified, so the database searches for private, public, and global database links. The database searches only for links that match the specified global database name.

Name Resolution When the Global Database Name Is Partial

If any part of the domain is specified, the database assumes that a complete global database name is specified. If a SQL statement specifies a partial global database name (that is, only the database component is specified), the database appends the value in the `DB_DOMAIN` initialization parameter to the value in the `DB_NAME`

initialization parameter to construct a complete name. For example, assume you issue the following statements:

```
CONNECT scott/tiger@locdb
SELECT * FROM scott.emp@orders;
```

If the network domain for `locdb` is `us.acme.com`, then the database appends this domain to `orders` to construct the complete global database name of `orders.us.acme.com`. The database searches for database links that match only the constructed global name. If a matching link is not found, the database returns an error and the SQL statement cannot execute.

Name Resolution When No Global Database Name Is Specified

If a global object name references an object in the local database and a database link name is *not* specified using the @ symbol, then the database automatically detects that the object is local and does not search for or use database links to resolve the object reference. For example, assume that you issue the following statements:

```
CONNECT scott/tiger@locdb
SELECT * from scott.emp;
```

Because the second statement does not specify a global database name using a database link connect string, the database does not search for database links.

Terminating the Search for Name Resolution

The database does not necessarily stop searching for matching database links when it finds the first match. The database must search for matching private, public, and network database links until it determines a complete path to the remote database (both a remote account and service name).

The first match determines the remote schema as illustrated in the following table:

User Operation	Database Response	Example
Do *not* specify the `CONNECT` clause	Uses a connected user database link	`CREATE DATABASE LINK k1 USING 'prod'`
Do specify the `CONNECT TO ... IDENTIFIED BY` clause	Uses a fixed user database link	`CREATE DATABASE LINK k2 CONNECT TO scott IDENTIFIED BY tiger USING 'prod'`

User Operation	Database Response	Example
Specify the `CONNECT TO CURRENT_USER` clause	Uses a current user database link	`CREATE DATABASE LINK k3 CONNECT TO CURRENT_USER USING 'prod'`
Do *not* specify the `USING` clause	Searches until it finds a link specifying a database string. If matching database links are found and a string is never identified, the database returns an error.	`CREATE DATABASE LINK k4 CONNECT TO CURRENT_USER`

After the database determines a complete path, it creates a remote session, assuming that an identical connection is not already open on behalf of the same local session. If a session already exists, the database reuses it.

Schema Object Name Resolution

After the local Oracle Database connects to the specified remote database on behalf of the local user that issued the SQL statement, object resolution continues as if the remote user had issued the associated SQL statement. The first match determines the remote schema according to the following rules:

Type of Link Specified	Location of Object Resolution
A fixed user database link	Schema specified in the link creation statement
A connected user database link	Connected user's remote schema
A current user database link	Current user's schema

If the database cannot find the object, then it checks public objects of the remote database. If it cannot resolve the object, then the established remote session remains but the SQL statement cannot execute and returns an error.

The following are examples of global object name resolution in a distributed database system. For all the following examples, assume that:

Example of Global Object Name Resolution: Complete Object Name

This example illustrates how the database resolves a complete global object name and determines the appropriate path to the remote database using both a private and public database link. For this example, assume the following:

- The remote database is named `sales.division3.acme.com`.
- The local database is named `hq.division3.acme.com`.
- A directory server (and therefore, global database links) is not available.
- A remote table `emp` is contained in the schema `tsmith`.

Consider the following statements issued by `scott` at the local database:

```
CONNECT scott/tiger@hq

CREATE PUBLIC DATABASE LINK sales.division3.acme.com
CONNECT TO guest IDENTIFIED BY network
  USING 'dbstring';
```

Later, JWARD connects and issues the following statements:

```
CONNECT jward/bronco@hq

CREATE DATABASE LINK sales.division3.acme.com
  CONNECT TO tsmith IDENTIFIED BY radio;

UPDATE tsmith.emp@sales.division3.acme.com
  SET deptno = 40
  WHERE deptno = 10;
```

The database processes the final statement as follows:

1. The database determines that a complete global object name is referenced in `jward`'s UPDATE statement. Therefore, the system begins searching in the local database for a database link with a matching name.
2. The database finds a matching private database link in the schema `jward`. Nevertheless, the private database link `jward.sales.division3.acme.com` does not indicate a complete path to the remote `sales` database, only a remote account. Therefore, the database now searches for a matching public database link.
3. The database finds the public database link in `scott`'s schema. From this public database link, the database takes the service name `dbstring`.

4. Combined with the remote account taken from the matching private fixed user database link, the database determines a complete path and proceeds to establish a connection to the remote `sales` database as user `tsmith/radio`.
5. The remote database can now resolve the object reference to the `emp` table. The database searches in the `tsmith` schema and finds the referenced `emp` table.
6. The remote database completes the execution of the statement and returns the results to the local database.

Example of Global Object Name Resolution: Partial Object Name

This example illustrates how the database resolves a partial global object name and determines the appropriate path to the remote database using both a private and public database link.

For this example, assume that:

- The remote database is named `sales.division3.acme.com`.
- The local database is named `hq.division3.acme.com`.
- A directory server (and therefore, global database links) is not available.
- A table `emp` on the remote database `sales` is contained in the schema `tsmith`, but not in schema `scott`.
- A public synonym named `emp` resides at remote database `sales` and points to `tsmith.emp` in the remote database `sales`.
- The public database link in "Example of Global Object Name Resolution: Complete Object Name" on page 29-40 is already created on local database `hq`:

```
CREATE PUBLIC DATABASE LINK sales.division3.acme.com
  CONNECT TO guest IDENTIFIED BY network
  USING 'dbstring';
```

Consider the following statements issued at local database `hq`:

```
CONNECT scott/tiger@hq

CREATE DATABASE LINK sales.division3.acme.com;

DELETE FROM emp@sales
  WHERE empno = 4299;
```

The database processes the final `DELETE` statement as follows:

1. The database notices that a partial global object name is referenced in `scott`'s `DELETE` statement. It expands it to a complete global object name using the domain of the local database as follows:

   ```
   DELETE FROM emp@sales.division3.acme.com
     WHERE empno = 4299;
   ```

2. The database searches the local database for a database link with a matching name.
3. The database finds a matching *private* connected user link in the schema `scott`, but the private database link indicates no path at all. The database uses the connected username/password as the remote account portion of the path and then searches for and finds a matching *public* database link:

   ```
   CREATE PUBLIC DATABASE LINK sales.division3.acme.com
     CONNECT TO guest IDENTIFIED BY network
     USING 'dbstring';
   ```

4. The database takes the database net service name `dbstring` from the public database link. At this point, the database has determined a complete path.
5. The database connects to the remote database as `scott/tiger` and searches for and does not find an object named `emp` in the schema `scott`.
6. The remote database searches for a public synonym named `emp` and finds it.
7. The remote database executes the statement and returns the results to the local database.

Global Name Resolution in Views, Synonyms, and Procedures

A view, synonym, or PL/SQL program unit (for example, a procedure, function, or trigger) can reference a remote schema object by its global object name. If the global object name is complete, then the database stores the definition of the object without expanding the global object name. If the name is partial, however, the database expands the name using the domain of the local database name.

The following table explains when the database completes the expansion of a partial global object name for views, synonyms, and program units:

User Operation	Database Response
Create a view	Does *not* expand partial global names. The data dictionary stores the exact text of the defining query. Instead, the database expands a partial global object name each time a statement that uses the view is parsed.
Create a synonym	Expands partial global names. The definition of the synonym stored in the data dictionary includes the expanded global object name.
Compile a program unit	Expands partial global names.

What Happens When Global Names Change

Global name changes can affect views, synonyms, and procedures that reference remote data using partial global object names. If the global name of the referenced database changes, views and procedures may try to reference a nonexistent or incorrect database. On the other hand, synonyms do not expand database link names at runtime, so they do not change.

Scenarios for Global Name Changes

For example, consider two databases named `sales.uk.acme.com` and `hq.uk.acme.com`. Also, assume that the `sales` database contains the following view and synonym:

```
CREATE VIEW employee_names AS
        SELECT ename FROM scott.emp@hr;

CREATE SYNONYM employee FOR scott.emp@hr;
```

The database expands the `employee` synonym definition and stores it as:

```
scott.emp@hr.uk.acme.com
```

Scenario 1: Both Databases Change Names First, consider the situation where both the Sales and Human Resources departments are relocated to the United States. Consequently, the corresponding global database names are both changed as follows:

- `sales.uk.acme.com` becomes `sales.us.acme.com`
- `hq.uk.acme.com` becomes `hq.us.acme.com`

The following table describes query expansion before and after the change in global names:

Query on sales	Expansion Before Change	Expansion After Change
SELECT * FROM employee_names	SELECT * FROM scott.emp@hr.uk.acme.com	SELECT * FROM scott.emp@hr.us.acme.com
SELECT * FROM employee	SELECT * FROM scott.emp@hr.uk.acme.com	SELECT * FROM scott.emp@hr.uk.acme.com

Scenario 2: One Database Changes Names Now consider that only the Sales department is moved to the United States; Human Resources remains in the UK. Consequently, the corresponding global database names are both changed as follows:

- `sales.uk.acme.com` becomes `sales.us.acme.com`
- `hq.uk.acme.com` is not changed

The following table describes query expansion before and after the change in global names:

Query on sales	Expansion Before Change	Expansion After Change
SELECT * FROM employee_names	SELECT * FROM scott.emp@hr.uk.acme.com	SELECT * FROM scott.emp@hr.us.acme.com
SELECT * FROM employee	SELECT * FROM scott.emp@hr.uk.acme.com	SELECT * FROM scott.emp@hr.uk.acme.com

In this case, the defining query of the `employee_names` view expands to a nonexistent global database name. On the other hand, the `employee` synonym continues to reference the correct database, `hq.uk.acme.com`.

Distributed Database Application Development

Application development in a distributed system raises issues that are not applicable in a nondistributed system. This section contains the following topics relevant for distributed application development:

- Transparency in a Distributed Database System
- Remote Procedure Calls (RPCs)
- Distributed Query Optimization

See Also: Chapter 31, "Developing Applications for a Distributed Database System" to learn how to develop applications for distributed systems

Transparency in a Distributed Database System

With minimal effort, you can develop applications that make an Oracle Database distributed database system transparent to users that work with the system. The goal of transparency is to make a distributed database system appear as though it is a single Oracle Database. Consequently, the system does not burden developers and users of the system with complexities that would otherwise make distributed database application development challenging and detract from user productivity.

The following sections explain more about transparency in a distributed database system.

Location Transparency

An Oracle Database distributed database system has features that allow application developers and administrators to hide the physical location of database objects from applications and users. **Location transparency** exists when a user can universally refer to a database object such as a table, regardless of the node to which an application connects. Location transparency has several benefits, including:

- Access to remote data is simple, because database users do not need to know the physical location of database objects.
- Administrators can move database objects with no impact on end-users or existing database applications.

Typically, administrators and developers use synonyms to establish location transparency for the tables and supporting objects in an application schema. For example, the following statements create synonyms in a database for tables in another, remote database.

```
CREATE PUBLIC SYNONYM emp
  FOR scott.emp@sales.us.americas.acme_auto.com;
CREATE PUBLIC SYNONYM dept
  FOR scott.dept@sales.us.americas.acme_auto.com;
```

Now, rather than access the remote tables with a query such as:

```
SELECT ename, dname
  FROM scott.emp@sales.us.americas.acme_auto.com e,
       scott.dept@sales.us.americas.acme_auto.com d
```

```
  WHERE e.deptno = d.deptno;
```

An application can issue a much simpler query that does not have to account for the location of the remote tables.

```
SELECT ename, dname
  FROM emp e, dept d
  WHERE e.deptno = d.deptno;
```

In addition to synonyms, developers can also use views and stored procedures to establish location transparency for applications that work in a distributed database system.

SQL and COMMIT Transparency

The Oracle Database distributed database architecture also provides query, update, and transaction transparency. For example, standard SQL statements such as `SELECT`, `INSERT`, `UPDATE`, and `DELETE` work just as they do in a nondistributed database environment. Additionally, applications control transactions using the standard SQL statements `COMMIT`, `SAVEPOINT`, and `ROLLBACK`. There is no requirement for complex programming or other special operations to provide distributed transaction control.

- The statements in a single transaction can reference any number of local or remote tables.
- The database guarantees that all nodes involved in a distributed transaction take the same action: they either all commit or all roll back the transaction.
- If a network or system failure occurs during the commit of a distributed transaction, the transaction is automatically and transparently resolved globally. Specifically, when the network or system is restored, the nodes either all commit or all roll back the transaction.

Internal to the database, each committed transaction has an associated **system change number** (**SCN**) to uniquely identify the changes made by the statements within that transaction. In a distributed database, the SCNs of communicating nodes are coordinated when:

- A connection is established using the path described by one or more database links.
- A distributed SQL statement is executed.
- A distributed transaction is committed.

Among other benefits, the coordination of SCNs among the nodes of a distributed database system allows global distributed read-consistency at both the statement and transaction level. If necessary, global distributed time-based recovery can also be completed.

Replication Transparency

The database also provide many features to transparently replicate data among the nodes of the system. For more information about Oracle Database replication features, see *Oracle Database Advanced Replication*.

Remote Procedure Calls (RPCs)

Developers can code PL/SQL packages and procedures to support applications that work with a distributed database. Applications can make local procedure calls to perform work at the local database and **remote procedure calls** (**RPCs**) to perform work at a remote database.

When a program calls a remote procedure, the local server passes all procedure parameters to the remote server in the call. For example, the following PL/SQL program unit calls the packaged procedure `del_emp` located at the remote `sales` database and passes it the parameter 1257:

```
BEGIN
 emp_mgmt.del_emp@sales.us.americas.acme_auto.com(1257);
END;
```

In order for the RPC to succeed, the called procedure must exist at the remote site, and the user being connected to must have the proper privileges to execute the procedure.

When developing packages and procedures for distributed database systems, developers must code with an understanding of what program units should do at remote locations, and how to return the results to a calling application.

Distributed Query Optimization

Distributed query optimization is an Oracle Database feature that reduces the amount of data transfer required between sites when a transaction retrieves data from remote tables referenced in a distributed SQL statement.

Distributed query optimization uses cost-based optimization to find or generate SQL expressions that extract only the necessary data from remote tables, process that data at a remote site or sometimes at the local site, and send the results to the

local site for final processing. This operation reduces the amount of required data transfer when compared to the time it takes to transfer all the table data to the local site for processing.

Using various cost-based optimizer hints such as `DRIVING_SITE`, `NO_MERGE`, and `INDEX`, you can control where Oracle Database processes the data and how it accesses the data.

> **See Also:** "Using Cost-Based Optimization" on page 31-4 for more information about cost-based optimization

Character Set Support for Distributed Environments

Oracle Database supports environments in which clients, Oracle Database servers, and non-Oracle Database servers use different character sets. `NCHAR` support is provided for heterogeneous environments. You can set a variety of National Language Support (NLS) and Heterogeneous Services (HS) environment variables and initialization parameters to control data conversion between different character sets.

Character settings are defined by the following NLS and HS parameters:

Parameters	Environment	Defined For
`NLS_LANG` (environment variable)	Client-Server	Client

`NLS_LANGUAGE`
`NLS_CHARACTERSET`
`NLS_TERRITORY`
	Client-Server
	Not Heterogeneous Distributed
	Heterogeneous Distributed
		Oracle Database server

`HS_LANGUAGE`	Heterogeneous Distributed	Non-Oracle Database server Transparent gateway
`NLS_NCHAR` (environment variable) `HS_NLS_NCHAR`	Heterogeneous Distributed	Oracle Database server Transparent gateway

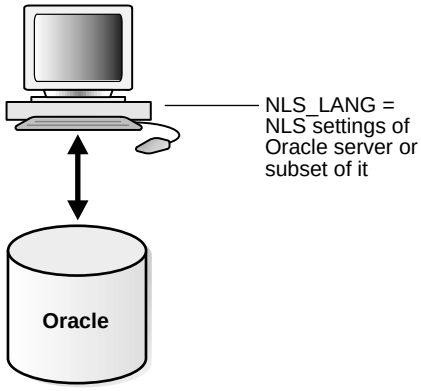
See Also:

- *Oracle Database Globalization Support Guide* for information about NLS parameters
- *Oracle Database Heterogeneous Connectivity Administrator's Guide* for information about HS parameters

Client/Server Environment

In a client / server environment, set the client character set to be the same as or a subset of the Oracle Database server character set, as illustrated in Figure 29–6:

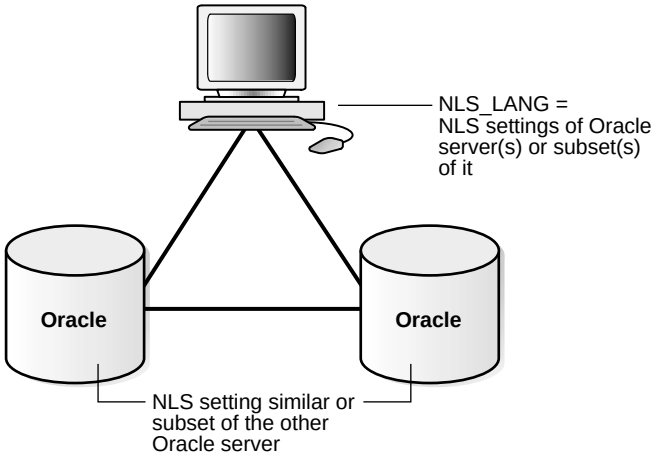
Figure 29–6 NLS Parameter Settings in a Client-Server Environment



Homogeneous Distributed Environment

In a nonheterogeneous environment, the client and server character sets should be either the same as or subsets of the main server character set, as illustrated in Figure 29–7:

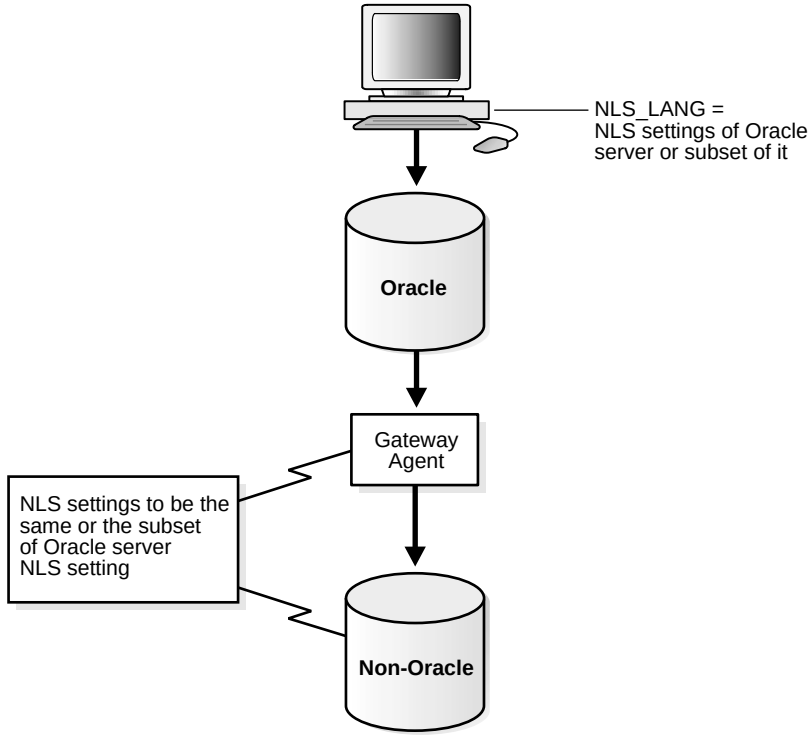
Figure 29–7 NLS Parameter Settings in a Homogeneous Environment



Heterogeneous Distributed Environment

In a heterogeneous environment, the NLS settings of the client, the transparent gateway, and the non-Oracle Database data source should be either the same or a subset of the database server character set as illustrated in Figure 29–8. Transparent gateways have full globalization support.

Figure 29–8 NLS Parameter Settings in a Heterogeneous Environment



In a heterogeneous environment, only transparent gateways built with HS technology support complete NCHAR capabilities. Whether a specific transparent gateway supports NCHAR depends on the non-Oracle Database data source it is targeting. For information on how a particular transparent gateway handles NCHAR support, consult the system-specific transparent gateway documentation.

> **See Also:** *Oracle Database Heterogeneous Connectivity Administrator's Guide* for more detailed information about Heterogeneous Services

30

Managing a Distributed Database

This chapter describes how to manage and maintain a distributed database system and contains the following topics:

Managing Global Names in a Distributed System

In a distributed database system, each database should have a unique **global database name**. Global database names uniquely identify a database in the system. A primary administration task in a distributed system is managing the creation and alteration of global database names.

This section contains the following topics:

Understanding How Global Database Names Are Formed

A global database name is formed from two components: a database name and a domain. The database name and the domain name are determined by the following initialization parameters at database creation:

Component	Parameter	Requirements	Example
Database name	DB_NAME	Must be eight characters or less.	sales
Domain containing the database	DB_DOMAIN	Must follow standard Internet conventions. Levels in domain names must be separated by dots and the order of domain names is from leaf to root, left to right.	us.acme.com

These are examples of valid global database names:

DB_NAME	DB_DOMAIN	Global Database Name
sales	au.oracle.com	sales.au.oracle.com
sales	us.oracle.com	sales.us.oracle.com
mktg	us.oracle.com	mktg.us.oracle.com
payroll	nonprofit.org	payroll.nonprofit.org

The DB_DOMAIN initialization parameter is only important at database creation time when it is used, together with the DB_NAME parameter, to form the database global name. At this point, the database global name is stored in the data dictionary. You must change the global name using an ALTER DATABASE statement, *not* by altering the DB_DOMAIN parameter in the initialization parameter file. It is good practice, however, to change the DB_DOMAIN parameter to reflect the change in the domain name before the next database startup.

Determining Whether Global Naming Is Enforced

The name that you give to a link on the local database depends on whether the remote database that you want to access enforces global naming. If the remote database enforces global naming, then you must use the remote database global database name as the name of the link. For example, if you are connected to the

local hq server and want to create a link to the remote mfg database, and mfg enforces global naming, then you must use the mfg global database name as the link name.

You can also use service names as part of the database link name. For example, if you use the service names sn1 and sn2 to connect to database hq.acme.com, and hq enforces global naming, then you can create the following link names to hq:

- HQ.ACME.COM@SN1
- HQ.ACME.COM@SN2

> **See Also:** "Using Connection Qualifiers to Specify Service Names Within Link Names" on page 30-13 for more information about using services names in link names

To determine whether global naming on a database is enforced on a database, either examine the database initialization parameter file or query the V$PARAMETER view. For example, to see whether global naming is enforced on mfg, you could start a session on mfg and then create and execute the following globalnames.sql script (sample output included):

```
COL NAME FORMAT A12
COL VALUE FORMAT A6
SELECT NAME, VALUE FROM V$PARAMETER
   WHERE NAME = 'global_names'
/

SQL> @globalnames

NAME         VALUE
------------ ------
global_names FALSE
```

Viewing a Global Database Name

Use the data dictionary view GLOBAL_NAME to view the database global name. For example, issue the following:

```
SELECT * FROM GLOBAL_NAME;

GLOBAL_NAME
--------------------------------------------------------------------------------
SALES.AU.ORACLE.COM
```

Changing the Domain in a Global Database Name

Use the ALTER DATABASE statement to change the domain in a database global name. Note that after the database is created, changing the initialization parameter DB_DOMAIN has no effect on the global database name or on the resolution of database link names.

The following example shows the syntax for the renaming statement, where *database* is a database name and *domain* is the network domain:

```
ALTER DATABASE RENAME GLOBAL_NAME TO database.domain;
```

Use the following procedure to change the domain in a global database name:

1. Determine the current global database name. For example, issue:

   ```
   SELECT * FROM GLOBAL_NAME;

   GLOBAL_NAME
   --------------------------------------------------------------------------------
   SALES.AU.ORACLE.COM
   ```

2. Rename the global database name using an ALTER DATABASE statement. For example, enter:

   ```
   ALTER DATABASE RENAME GLOBAL_NAME TO sales.us.oracle.com;
   ```

3. Query the GLOBAL_NAME table to check the new name. For example, enter:

   ```
   SELECT * FROM GLOBAL_NAME;

   GLOBAL_NAME
   --------------------------------------------------------------------------------
   SALES.US.ORACLE.COM
   ```

Changing a Global Database Name: Scenario

In this scenario, you change the domain part of the global database name of the local database. You also create database links using partially specified global names to test how Oracle Database resolves the names. You discover that the database resolves the partial names using the domain part of the current global database name of the local database, not the value for the initialization parameter DB_DOMAIN.

1. You connect to SALES.US.ACME.COM and query the GLOBAL_NAME data dictionary view to determine the current database global name:

```
CONNECT SYSTEM/password@sales.us.acme.com
SELECT * FROM GLOBAL_NAME;

GLOBAL_NAME
--------------------------------------------------------------------------------
SALES.US.ACME.COM
```

2. You query the V$PARAMETER view to determine the current setting for the DB_DOMAIN initialization parameter:

```
SELECT NAME, VALUE FROM V$PARAMETER WHERE NAME = 'db_domain';

NAME       VALUE
---------  -----------
db_domain  US.ACME.COM
```

3. You then create a database link to a database called hq, using only a partially-specified global name:

```
CREATE DATABASE LINK hq USING 'sales';
```

   The database expands the global database name for this link by appending the domain part of the global database name of the *local* database to the name of the database specified in the link.

4. You query USER_DB_LINKS to determine which domain name the database uses to resolve the partially specified global database name:

```
SELECT DB_LINK FROM USER_DB_LINKS;

DB_LINK
------------------
HQ.US.ACME.COM
```

   This result indicates that the domain part of the global database name of the local database is us.acme.com. The database uses this domain in resolving partial database link names when the database link is created.

5. Because you have received word that the sales database will move to Japan, you rename the sales database to sales.jp.acme.com:

```
ALTER DATABASE RENAME GLOBAL_NAME TO sales.jp.acme.com;
SELECT * FROM GLOBAL_NAME;

GLOBAL_NAME
--------------------------------------------------------------------------------
```

```
SALES.JP.ACME.COM
```

6. You query V$PARAMETER again and discover that the value of DB_DOMAIN is *not* changed, although you renamed the domain part of the global database name:

```
SELECT NAME, VALUE FROM V$PARAMETER
  WHERE NAME = 'db_domain';

NAME       VALUE
---------  -----------
db_domain  US.ACME.COM
```

   This result indicates that the value of the DB_DOMAIN initialization parameter is independent of the ALTER DATABASE RENAME GLOBAL_NAME statement. The ALTER DATABASE statement determines the domain of the global database name, not the DB_DOMAIN initialization parameter (although it is good practice to alter DB_DOMAIN to reflect the new domain name).

7. You create another database link to database supply, and then query USER_DB_LINKS to see how the database resolves the domain part of the global database name of supply:

```
CREATE DATABASE LINK supply USING 'supply';
SELECT DB_LINK FROM USER_DB_LINKS;

DB_LINK
------------------
HQ.US.ACME.COM
SUPPLY.JP.ACME.COM
```

   This result indicates that the database resolves the partially specified link name by using the domain jp.acme.com. This domain is used when the link is created because it is the domain part of the global database name of the local database. The database does *not* use the DB_DOMAIN initialization parameter setting when resolving the partial link name.

8. You then receive word that your previous information was faulty: sales will be in the ASIA.JP.ACME.COM domain, not the JP.ACME.COM domain. Consequently, you rename the global database name as follows:

```
ALTER DATABASE RENAME GLOBAL_NAME TO sales.asia.jp.acme.com;
SELECT * FROM GLOBAL_NAME;

GLOBAL_NAME
--------------------------------------------------------------------------------
```

```
SALES.ASIA.JP.ACME.COM
```

9. You query V$PARAMETER to again check the setting for the parameter DB_DOMAIN:

```
SELECT NAME, VALUE FROM V$PARAMETER
  WHERE NAME = 'db_domain';

NAME        VALUE
----------  -----------
db_domain   US.ACME.COM
```

   The result indicates that the domain setting in the parameter file is exactly the same as it was before you issued *either* of the ALTER DATABASE RENAME statements.

10. Finally, you create a link to the warehouse database and again query USER_DB_LINKS to determine how the database resolves the partially-specified global name:

```
CREATE DATABASE LINK warehouse USING 'warehouse';
SELECT DB_LINK FROM USER_DB_LINKS;

DB_LINK
------------------
HQ.US.ACME.COM
SUPPLY.JP.ACME.COM
WAREHOUSE.ASIA.JP.ACME.COM
```

   Again, you see that the database uses the domain part of the global database name of the local database to expand the partial link name during link creation.

Note: In order to correct the supply database link, it must be dropped and re-created.

See Also: *Oracle Database Reference* for more information about specifying the DB_NAME and DB_DOMAIN initialization parameters

Creating Database Links

To support application access to the data and schema objects throughout a distributed database system, you must create all necessary database links. This section contains the following topics:

- Obtaining Privileges Necessary for Creating Database Links
- Specifying Link Types
- Specifying Link Users
- Using Connection Qualifiers to Specify Service Names Within Link Names

Obtaining Privileges Necessary for Creating Database Links

A database link is a pointer in the local database that lets you access objects on a remote database. To create a private database link, you must have been granted the proper privileges. The following table illustrates which privileges are required on which database for which type of link:

Privilege	Database	Required For
CREATE DATABASE LINK	Local	Creation of a private database link.
CREATE PUBLIC DATABASE LINK	Local	Creation of a public database link.
CREATE SESSION	Remote	Creation of any type of database link.

To see which privileges you currently have available, query ROLE_SYS_PRIVS. For example, you could create and execute the following privs.sql script (sample output included):

```
SELECT DISTINCT PRIVILEGE AS "Database Link Privileges"
FROM ROLE_SYS_PRIVS
WHERE PRIVILEGE IN ( 'CREATE SESSION','CREATE DATABASE LINK',
                     'CREATE PUBLIC DATABASE LINK')
/

SQL> @privs

Database Link Privileges
----------------------------------------
CREATE DATABASE LINK
CREATE PUBLIC DATABASE LINK
CREATE SESSION
```

Specifying Link Types

When you create a database link, you must decide who will have access to it. The following sections describe how to create the three basic types of links:

- Creating Private Database Links
- Creating Public Database Links
- Creating Global Database Links

Creating Private Database Links

To create a private database link, specify the following (where *link_name* is the global database name or an arbitrary link name):

```
CREATE DATABASE LINK link_name ...;
```

Following are examples of private database links:

SQL Statement	Result
`CREATE DATABASE LINK supply.us.acme.com;`	A private link using the global database name to the remote `supply` database. The link uses the userid / password of the connected user. So if `scott` (identified by `tiger`) uses the link in a query, the link establishes a connection to the remote database as `scott/tiger`.
`CREATE DATABASE LINK link_2 CONNECT TO jane IDENTIFIED BY doe USING 'us_supply';`	A private fixed user link called link_2 to the database with service name us_supply. The link connects to the remote database with the userid / password of jane / doe regardless of the connected user.
`CREATE DATABASE LINK link_1 CONNECT TO CURRENT_USER USING 'us_supply';`	A private link called `link_1` to the database with service name `us_supply`. The link uses the userid / password of the current user to log onto the remote database. **Note:** The current user may not be the same as the connected user, and must be a global user on both databases involved in the link (see "Users of Database Links" on page 29-16). Current user links are part of the Oracle Advanced Security option.

See Also: *Oracle Database SQL Reference* for `CREATE DATABASE LINK` syntax

Creating Public Database Links

To create a public database link, use the keyword `PUBLIC` (where *link_name* is the global database name or an arbitrary link name):

```
CREATE PUBLIC DATABASE LINK link_name ...;
```

Following are examples of public database links:

SQL Statement	Result
`CREATE PUBLIC DATABASE LINK supply.us.acme.com;`	A public link to the remote `supply` database. The link uses the userid / password of the connected user. So if `scott` (identified by `tiger`) uses the link in a query, the link establishes a connection to the remote database as `scott/tiger`.
`CREATE PUBLIC DATABASE LINK pu_link CONNECT TO CURRENT_ USER USING 'supply';`	A public link called `pu_link` to the database with service name `supply`. The link uses the userid / password of the current user to log onto the remote database. **Note:** The current user may not be the same as the connected user, and must be a global user on both databases involved in the link (see "Users of Database Links" on page 29-16).
`CREATE PUBLIC DATABASE LINK sales.us.acme.com CONNECT TO jane IDENTIFIED BY doe;`	A public fixed user link to the remote `sales` database. The link connects to the remote database with the userid / password of `jane/doe`.

See Also: *Oracle Database SQL Reference* for `CREATE PUBLIC DATABASE LINK` syntax

Creating Global Database Links

In earlier releases, you defined **global database links** in the Oracle Names server. The Oracle Names server has been deprecated. Now, you can use a directory server in which databases are identified by net service names. In this document these are what are referred to as global database links.

See the *Oracle Net Services Administrator's Guide* to learn how to create directory entries that act as global database links.

Specifying Link Users

A database link defines a communication path from one database to another. When an application uses a database link to access a remote database, Oracle Database establishes a database session in the remote database on behalf of the local application request.

When you create a private or public database link, you can determine which schema on the remote database the link will establish connections to by creating fixed user, current user, and connected user database links.

Creating Fixed User Database Links

To create a **fixed user database link**, you embed the credentials (in this case, a username and password) required to access the remote database in the definition of the link:

```
CREATE DATABASE LINK ... CONNECT TO username IDENTIFIED BY password ...;
```

Following are examples of fixed user database links:

SQL Statement	Result
`CREATE PUBLIC DATABASE LINK supply.us.acme.com CONNECT TO scott AS tiger;`	A public link using the global database name to the remote `supply` database. The link connects to the remote database with the userid / password `scott/tiger`.
`CREATE DATABASE LINK foo CONNECT TO jane IDENTIFIED BY doe USING 'finance';`	A private fixed user link called `foo` to the database with service name `finance`. The link connects to the remote database with the userid / password `jane/doe`.

When an application uses a fixed user database link, the local server always establishes a connection to a fixed remote schema in the remote database. The local server also sends the fixed user's credentials across the network when an application uses the link to access the remote database.

Creating Connected User and Current User Database Links

Connected user and current user database links do not include credentials in the definition of the link. The credentials used to connect to the remote database can change depending on the user that references the database link and the operation performed by the application.

> **Note:** For many distributed applications, you do not want a user to have privileges in a remote database. One simple way to achieve this result is to create a procedure that contains a fixed user or current user database link within it. In this way, the user accessing the procedure temporarily assumes someone else's privileges.

For an extended conceptual discussion of the distinction between connected users and current users, see "Users of Database Links" on page 29-16.

Creating a Connected User Database Link To create a connected user database link, omit the `CONNECT TO` clause. The following syntax creates a connected user database link, where *dblink* is the name of the link and *net_service_name* is an optional connect string:

```
CREATE [SHARED] [PUBLIC] DATABASE LINK dblink ... [USING 'net_service_name'];
```

For example, to create a connected user database link, use the following syntax:

```
CREATE DATABASE LINK sales.division3.acme.com USING 'sales';
```

Creating a Current User Database Link To create a current user database link, use the `CONNECT TO CURRENT_USER` clause in the link creation statement. Current user links are only available through the Oracle Advanced Security option.

The following syntax creates a current user database link, where *dblink* is the name of the link and *net_service_name* is an optional connect string:

```
CREATE [SHARED] [PUBLIC] DATABASE LINK dblink CONNECT TO CURRENT_USER
[USING 'net_service_name'];
```

For example, to create a connected user database link to the `sales` database, you might use the following syntax:

```
CREATE DATABASE LINK sales CONNECT TO CURRENT_USER USING 'sales';
```

> **Note:** To use a current user database link, the current user must be a global user on both databases involved in the link.

> **See Also:** *Oracle Database SQL Reference* for more syntax information about creating database links

Using Connection Qualifiers to Specify Service Names Within Link Names

In some situations, you may want to have several database links of the same type (for example, public) that point to the same remote database, yet establish connections to the remote database using different communication pathways. Some cases in which this strategy is useful are:

- A remote database is part of an Oracle Real Application Clusters configuration, so you define several public database links at your local node so that connections can be established to specific instances of the remote database.
- Some clients connect to the Oracle Database server using TCP/IP while others use DECNET.

To facilitate such functionality, the database lets you create a database link with an optional service name in the database link name. When creating a database link, a service name is specified as the trailing portion of the database link name, separated by an @ sign, as in `@sales`. This string is called a **connection qualifier**.

For example, assume that remote database `hq.acme.com` is managed in a Oracle Real Application Clusters environment. The `hq` database has two instances named `hq_1` and `hq_2`. The local database can contain the following public database links to define pathways to the remote instances of the `hq` database:

```
CREATE PUBLIC DATABASE LINK hq.acme.com@hq_1
  USING 'string_to_hq_1';
CREATE PUBLIC DATABASE LINK hq.acme.com@hq_2
  USING 'string_to_hq_2';
CREATE PUBLIC DATABASE LINK hq.acme.com
  USING 'string_to_hq';
```

Notice in the first two examples that a service name is simply a part of the database link name. The text of the service name does not necessarily indicate how a connection is to be established; this information is specified in the service name of the `USING` clause. Also notice that in the third example, a service name is not specified as part of the link name. In this case, just as when a service name is specified as part of the link name, the instance is determined by the `USING` string.

To use a service name to specify a particular instance, include the service name at the end of the global object name:

```
SELECT * FROM scott.emp@hq.acme.com@hq_1
```

Note that in this example, there are two @ symbols.

Using Shared Database Links

Every application that references a remote server using a standard database link establishes a connection between the local database and the remote database. Many users running applications simultaneously can cause a high number of connections between the local and remote databases.

Shared database links enable you to limit the number of network connections required between the local server and the remote server.

This section contains the following topics:

- Determining Whether to Use Shared Database Links
- Creating Shared Database Links
- Configuring Shared Database Links

> **See Also:** "What Are Shared Database Links?" on page 29-10 for a conceptual overview of shared database links

Determining Whether to Use Shared Database Links

Look carefully at your application and shared server configuration to determine whether to use shared links. A simple guideline is to use shared database links when the number of users accessing a database link is expected to be much larger than the number of server processes in the local database.

The following table illustrates three possible configurations involving database links:

Link Type	Server Mode	Consequences
Nonshared	Dedicated / shared server	If your application uses a standard public database link, and 100 users simultaneously require a connection, then 100 direct network connections to the remote database are required.
Shared	Shared server	If 10 shared server processes exist in the local shared server mode database, then 100 users that use the same database link require 10 or fewer network connections to the remote server. Each local shared server process may only need one connection to the remote server.

Link Type	Server Mode	Consequences
Shared	Dedicated	If 10 clients connect to a local dedicated server, and each client has 10 sessions on the same connection (thus establishing 100 sessions overall), and each session references the same remote database, then only 10 connections are needed. With a nonshared database link, 100 connections are needed.

Shared database links are not useful in all situations. Assume that only one user accesses the remote server. If this user defines a shared database link and 10 shared server processes exist in the local database, then this user can require up to 10 network connections to the remote server. Because the user can use each shared server process, each process can establish a connection to the remote server.

Clearly, a nonshared database link is preferable in this situation because it requires only one network connection. Shared database links lead to more network connections in single-user scenarios, so use shared links only when many users need to use the same link. Typically, shared links are used for public database links, but can also be used for private database links when many clients access the same local schema (and therefore the same private database link).

> **Note:** In a multitiered environment, there is a restriction that if you use a shared database link to connect to a remote database, then that remote database cannot link to another database with a database link that cannot be migrated. That link must use a shared server, or it must be another shared database link.

Creating Shared Database Links

To create a shared database link, use the keyword SHARED in the CREATE DATABASE LINK statement:

```
CREATE SHARED DATABASE LINK dblink_name
[CONNECT TO username IDENTIFIED BY password]|[CONNECT TO CURRENT_USER]
AUTHENTICATED BY schema_name IDENTIFIED BY password
[USING 'service_name'];
```

The following example creates a fixed user, shared link to database `sales`, connecting as `scott` and authenticated as `keith`:

```
CREATE SHARED DATABASE LINK link2sales
CONNECT TO scott IDENTIFIED BY tiger
```

```
AUTHENTICATED BY keith IDENTIFIED BY richards
USING 'sales';
```

Whenever you use the keyword SHARED, the clause AUTHENTICATED BY is required. The schema specified in the AUTHENTICATED BY clause is only used for security reasons and can be considered a dummy schema. It is not affected when using shared database links, nor does it affect the users of the shared database link. The AUTHENTICATED BY clause is required to prevent unauthorized clients from masquerading as a database link user and gaining access to privileged information.

> **See Also:** *Oracle Database SQL Reference* for information about the CREATE DATABASE LINK statement

Configuring Shared Database Links

You can configure shared database links in the following ways:

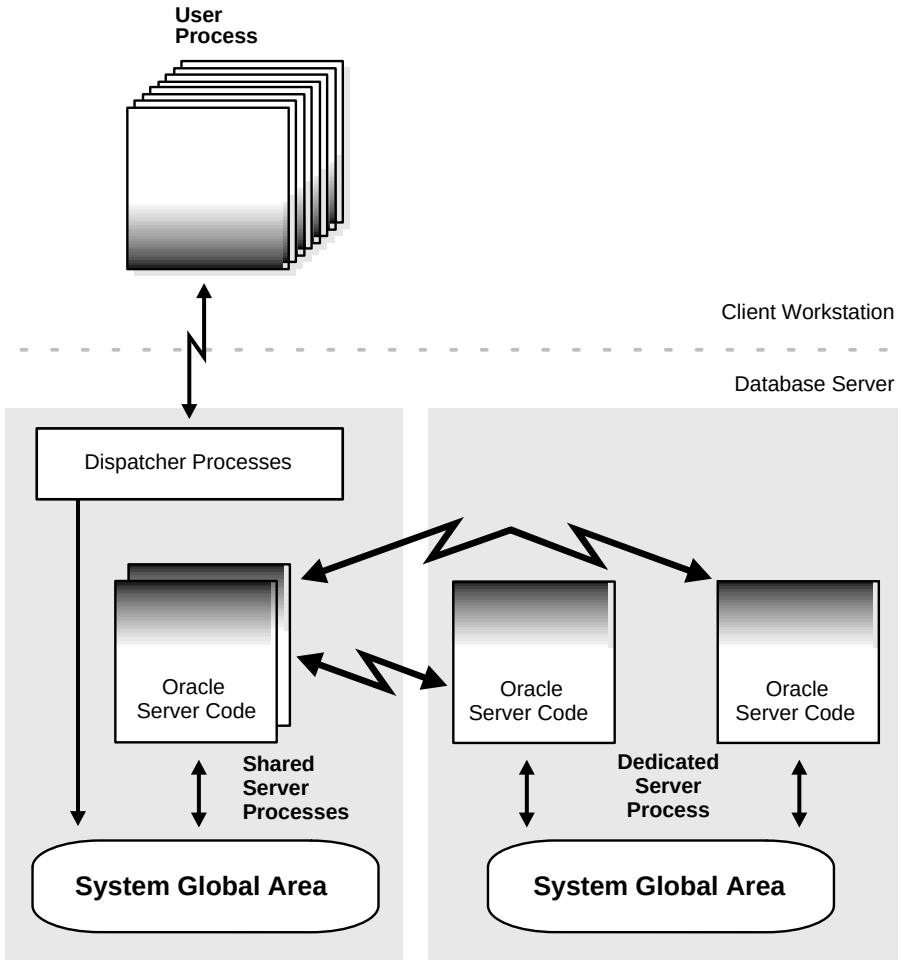
- Creating Shared Links to Dedicated Servers
- Creating Shared Links to Shared Servers

Creating Shared Links to Dedicated Servers

In the configuration illustrated in Figure 30–1, a shared server process in the local server owns a dedicated remote server process. The advantage is that a direct network transport exists between the local shared server and the remote dedicated server. A disadvantage is that extra back-end server processes are needed.

> **Note:** The remote server can either be a shared server or dedicated server. There is a dedicated connection between the local and remote servers. When the remote server is a shared server, you can force a dedicated server connection by using the (SERVER=DEDICATED) clause in the definition of the service name.

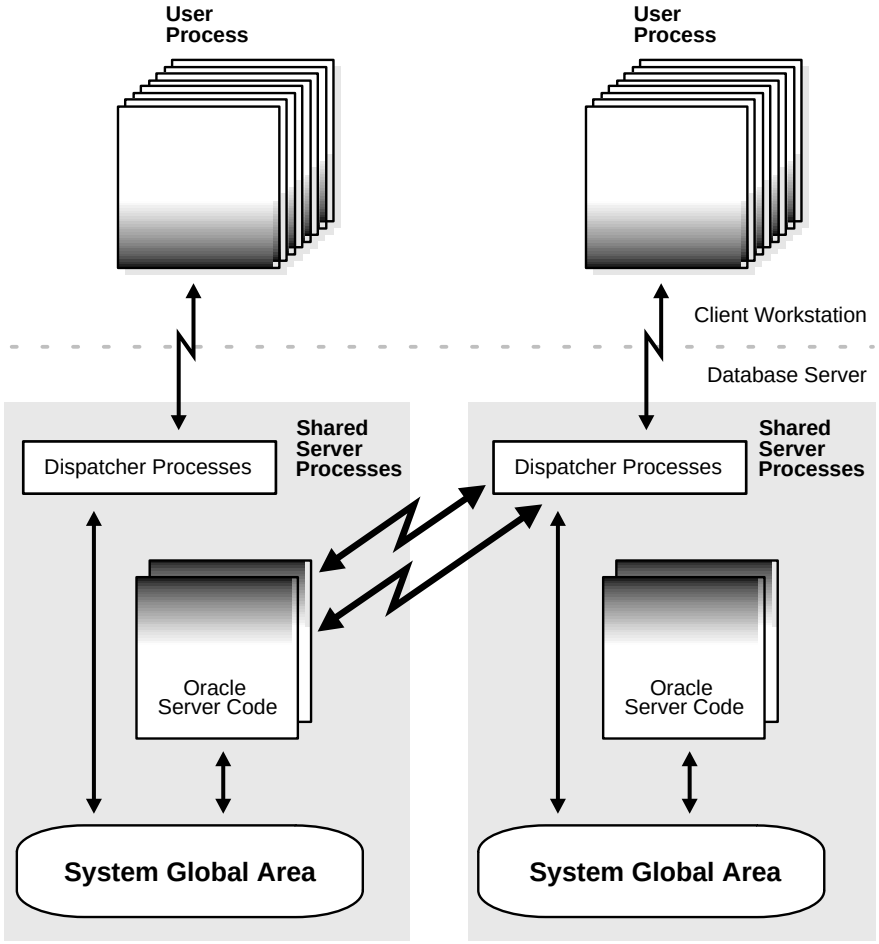
Figure 30–1 A Shared Database Link to Dedicated Server Processes



Creating Shared Links to Shared Servers

The configuration illustrated in Figure 30–2 uses shared server processes on the remote server. This configuration eliminates the need for more dedicated servers, but requires the connection to go through the dispatcher on the remote server. Note that both the local and the remote server must be configured as shared servers.

Figure 30–2 Shared Database Link to Shared Server



See Also: *Oracle Net Services Administrator's Guide* for information about the shared server option

Managing Database Links

This section contains the following topics:

- Closing Database Links
- Dropping Database Links

- Limiting the Number of Active Database Link Connections

Closing Database Links

If you access a database link in a session, then the link remains open until you close the session. A link is open in the sense that a process is active on each of the remote databases accessed through the link. This situation has the following consequences:

- If 20 users open sessions and access the same public link in a local database, then 20 database link connections are open.
- If 20 users open sessions and each user accesses a private link, then 20 database link connections are open.
- If one user starts a session and accesses 20 different links, then 20 database link connections are open.

After you close a session, the links that were active in the session are automatically closed. You may have occasion to close the link manually. For example, close links when:

- The network connection established by a link is used infrequently in an application.
- The user session must be terminated.

If you want to close a link, issue the following statement, where *linkname* refers to the name of the link:

```
ALTER SESSION CLOSE DATABASE LINK linkname;
```

Note that this statement only closes the links that are active in your current session.

Dropping Database Links

You can drop a database link just as you can drop a table or view. If the link is private, then it must be in your schema. If the link is public, then you must have the `DROP PUBLIC DATABASE LINK` system privilege.

The statement syntax is as follows, where *dblink* is the name of the link:

```
DROP [PUBLIC] DATABASE LINK dblink;
```

Procedure for Dropping a Private Database Link

1. Connect to the local database using SQL*Plus. For example, enter:

```
CONNECT scott/tiger@local_db
```

2. Query USER_DB_LINKS to view the links that you own. For example, enter:

```
SELECT DB_LINK FROM USER_DB_LINKS;

DB_LINK
----------------------------------
SALES.US.ORACLE.COM
MKTG.US.ORACLE.COM
2 rows selected.
```

3. Drop the desired link using the DROP DATABASE LINK statement. For example, enter:

```
DROP DATABASE LINK sales.us.oracle.com;
```

Procedure for Dropping a Public Database Link

1. Connect to the local database as a user with the DROP PUBLIC DATABASE LINK privilege. For example, enter:

```
CONNECT SYSTEM/password@local_db AS SYSDBA
```

2. Query DBA_DB_LINKS to view the public links. For example, enter:

```
SELECT DB_LINK FROM USER_DB_LINKS
  WHERE OWNER = 'PUBLIC';

DB_LINK
----------------------------------
DBL1.US.ORACLE.COM
SALES.US.ORACLE.COM
INST2.US.ORACLE.COM
RMAN2.US.ORACLE.COM
4 rows selected.
```

3. Drop the desired link using the DROP PUBLIC DATABASE LINK statement. For example, enter:

```
DROP PUBLIC DATABASE LINK sales.us.oracle.com;
```

Limiting the Number of Active Database Link Connections

You can limit the number of connections from a user process to remote databases using the static initialization parameter OPEN_LINKS. This parameter controls the

number of remote connections that a single user session can use concurrently in distributed transactions.

Note the following considerations for setting this parameter:

- The value should be greater than or equal to the number of databases referred to in a single SQL statement that references multiple databases.
- Increase the value if several distributed databases are accessed over time. Thus, if you regularly access three databases, set `OPEN_LINKS` to 3 or greater.
- The default value for `OPEN_LINKS` is 4. If `OPEN_LINKS` is set to 0, then no distributed transactions are allowed.

> **See Also:** *Oracle Database Reference* for more information about the `OPEN_LINKS` initialization parameter

Viewing Information About Database Links

The data dictionary of each database stores the definitions of all the database links in the database. You can use data dictionary tables and views to gain information about the links. This section contains the following topics:

- Determining Which Links Are in the Database
- Determining Which Link Connections Are Open

Determining Which Links Are in the Database

The following views show the database links that have been defined at the local database and stored in the data dictionary:

View	Purpose
`DBA_DB_LINKS`	Lists all database links in the database.
`ALL_DB_LINKS`	Lists all database links accessible to the connected user.
`USER_DB_LINKS`	Lists all database links owned by the connected user.

These data dictionary views contain the same basic information about database links, with some exceptions:

Column	Which Views?	Description
OWNER	All except USER_*	The user who created the database link. If the link is public, then the user is listed as PUBLIC.
DB_LINK	All	The name of the database link.
USERNAME	All	If the link definition includes a fixed user, then this column displays the username of the fixed user. If there is no fixed user, the column is NULL.
PASSWORD	Only USER_*	The password for logging into the remote database.
HOST	All	The net service name used to connect to the remote database.
CREATED	All	Creation time of the database link.

Any user can query USER_DB_LINKS to determine which database links are available to that user. Only those with additional privileges can use the ALL_DB_LINKS or DBA_DB_LINKS view.

The following script queries the DBA_DB_LINKS view to access link information:

```
COL OWNER FORMAT a10
COL USERNAME FORMAT A8 HEADING "USER"
COL DB_LINK FORMAT A30
COL HOST FORMAT A7 HEADING "SERVICE"
SELECT * FROM DBA_DB_LINKS
/
```

Here, the script is invoked and the resulting output is shown:

```
SQL>@link_script

OWNER      DB_LINK                        USER     SERVICE CREATED
---------- ------------------------------ -------- ------- ----------
SYS        TARGET.US.ACME.COM             SYS      inst1   23-JUN-99
PUBLIC     DBL1.UK.ACME.COM               BLAKE    ora51   23-JUN-99
PUBLIC     RMAN2.US.ACME.COM                       inst2   23-JUN-99
PUBLIC     DEPT.US.ACME.COM                        inst2   23-JUN-99
JANE       DBL.UK.ACME.COM                BLAKE    ora51   23-JUN-99
SCOTT      EMP.US.ACME.COM                SCOTT    inst2   23-JUN-99
6 rows selected.
```

Authorization for Viewing Password Information

Only USER_DB_LINKS contains a column for password information. However, if you are an administrative user (SYS or users who connect AS SYSDBA), then you

can view passwords for all links in the database by querying the LINK$ table. If you are not an administrative user, you can be authorized to query the LINK$ table by one of the following methods:

- Being granted specific object privilege for the LINK$ table
- Being granted the SELECT ANY DICTIONARY system privilege

> **See Also:** *Oracle Database Security Guide* for more information about privileges necessary to view objects in the SYS schema

Viewing Password Information

You can create and run the following script in SQL*Plus to obtain password information (sample output included):

```
COL USERID FORMAT A10
COL PASSWORD FORMAT A10
SELECT USERID,PASSWORD
  FROM SYS.LINK$
  WHERE PASSWORD IS NOT NULL
/

SQL>@linkpwd

USERID     PASSWORD
---------- ----------
SYS        ORACLE
BLAKE      TYGER
SCOTT      TIGER
3 rows selected.
```

Viewing Authentication Passwords

It is possible to view AUTHENTICATED BY ... IDENTIFIED BY ... usernames and passwords for all links in the database by querying the LINK$ table. You can create and run the following script in SQL*Plus to obtain password information (sample output included):

```
COL AUTHUSR FORMAT A10
COL AUTHPWD FORMAT A10
SELECT AUTHUSR AS userid, AUTHPWD AS password
  FROM SYS.LINK$
  WHERE PASSWORD IS NOT NULL
/
```

```
SQL> @authpwd

USERID     PASSWORD
---------- ----------
ELLIE      MAY
1 row selected.
```

You can also view the link and password information together in a join by creating and executing the following script (sample output included):

```
COL OWNER FORMAT A8
COL DB_LINK FORMAT A15
COL USERNAME FORMAT A8 HEADING "CON_USER"
COL PASSWORD FORMAT A8 HEADING "CON_PWD"
COL AUTHUSR FORMAT A8 HEADING "AUTH_USER"
COL AUTHPWD FORMAT A8 HEADING "AUTH_PWD"
COL HOST FORMAT A7 HEADING "SERVICE"
COL CREATED FORMAT A10

SELECT DISTINCT d.OWNER,d.DB_LINK,d.USERNAME,l.PASSWORD,
                l.AUTHUSR,l.AUTHPWD,d.HOST,d.CREATED
FROM DBA_DB_LINKS d, SYS.LINK$ l
WHERE PASSWORD IS NOT NULL
AND d.USERNAME = l.USERID
/

SQL> @user_and_pwd

OWNER    DB_LINK         CON_USER CON_PWD  AUTH_USE AUTH_PWD SERVICE CREATED
-------- --------------- -------- -------- -------- -------- ------- ----------
JANE     DBL.ACME.COM    BLAKE    TYGER    ELLIE    MAY      ora51   23-JUN-99
PUBLIC   DBL1.ACME.COM   SCOTT    TIGER                      ora51   23-JUN-99
SYS      TARGET.ACME.COM SYS      ORACLE                     inst1   23-JUN-99
```

Determining Which Link Connections Are Open

You may find it useful to determine which database link connections are currently open in your session. Note that if you connect as SYSDBA, you cannot query a view to determine all the links open for all sessions; you can only access the link information in the session within which you are working.

The following views show the database link connections that are currently open in your current session:

View	Purpose
V$DBLINK	Lists all open database links in your session, that is, all database links with the IN_TRANSACTION column set to YES.
GV$DBLINK	Lists all open database links in your session along with their corresponding instances. This view is useful in an Oracle Real Application Clusters configuration.

These data dictionary views contain the same basic information about database links, with one exception:

Column	Which Views?	Description
DB_LINK	All	The name of the database link.
OWNER_ID	All	The owner of the database link.
LOGGED_ON	All	Whether the database link is currently logged on.
HETEROGENEOUS	All	Whether the database link is homogeneous (NO) or heterogeneous (YES).
PROTOCOL	All	The communication protocol for the database link.
OPEN_CURSORS	All	Whether cursors are open for the database link.
IN_TRANSACTION	All	Whether the database link is accessed in a transaction that has not yet been committed or rolled back.
UPDATE_SENT	All	Whether there was an update on the database link.
COMMIT_POINT_ STRENGTH	All	The commit point strength of the transactions using the database link.
INST_ID	GV$DBLINK only	The instance from which the view information was obtained.

For example, you can create and execute the script below to determine which links are open (sample output included):

```
COL DB_LINK FORMAT A25
COL OWNER_ID FORMAT 99999 HEADING "OWNID"
COL LOGGED_ON FORMAT A5 HEADING "LOGON"
COL HETEROGENEOUS FORMAT A5 HEADING "HETER"
COL PROTOCOL FORMAT A8
COL OPEN_CURSORS FORMAT 999 HEADING "OPN_CUR"
```

```
COL IN_TRANSACTION FORMAT A3 HEADING "TXN"
COL UPDATE_SENT FORMAT A6 HEADING "UPDATE"
COL COMMIT_POINT_STRENGTH FORMAT 99999 HEADING "C_P_S"

SELECT * FROM V$DBLINK
/

SQL> @dblink

DB_LINK                   OWNID LOGON HETER PROTOCOL OPN_CUR TXN UPDATE  C_P_S
------------------------- ------ ----- ----- -------- ------- --- ------ ------
INST2.ACME.COM                 0 YES   YES    UNKN           0 YES YES       255
```

Creating Location Transparency

After you have configured the necessary database links, you can use various tools to hide the distributed nature of the database system from users. In other words, users can access remote objects as if they were local objects. The following sections explain how to hide distributed functionality from users:

- Using Views to Create Location Transparency
- Using Synonyms to Create Location Transparency
- Using Procedures to Create Location Transparency

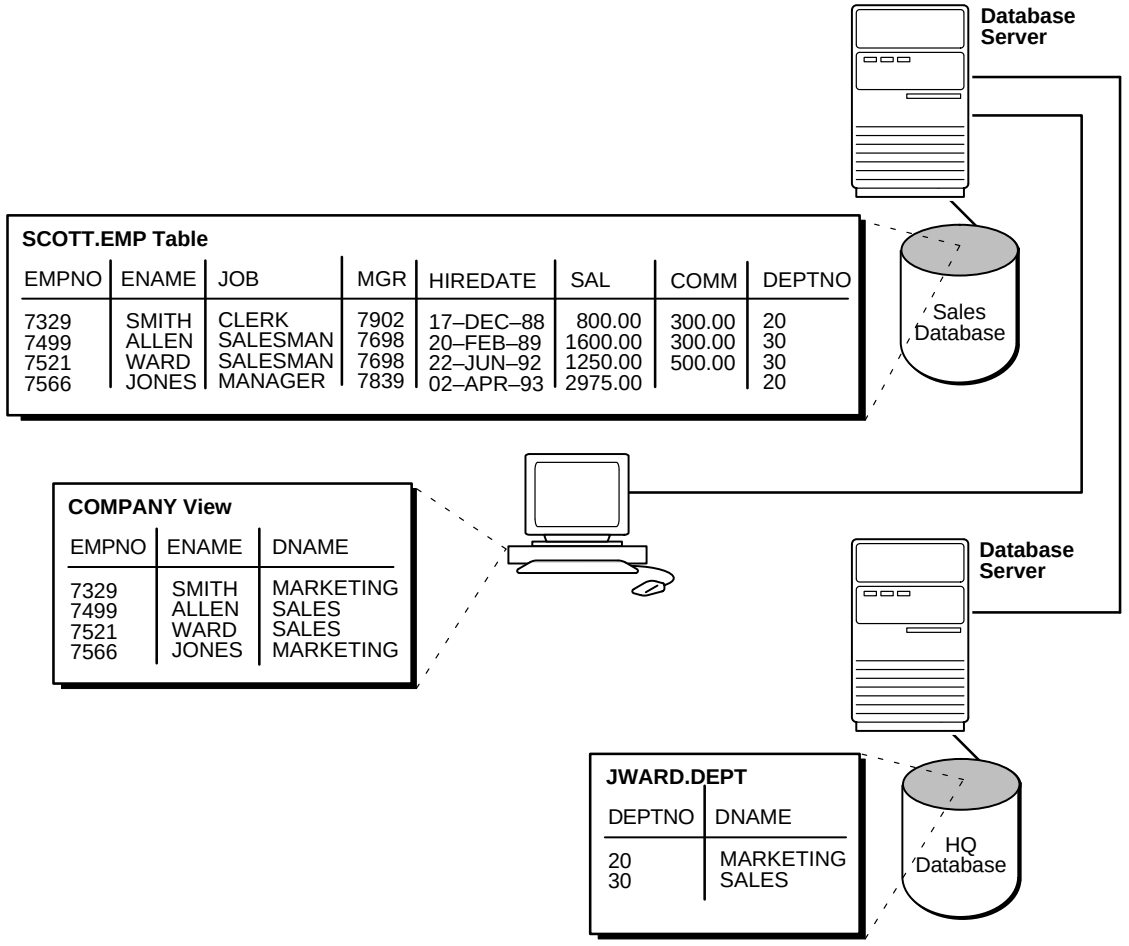
Using Views to Create Location Transparency

Local views can provide location transparency for local and remote tables in a distributed database system.

For example, assume that table `emp` is stored in a local database and table `dept` is stored in a remote database. To make these tables transparent to users of the system, you can create a view in the local database that joins local and remote data:

```
CREATE VIEW company AS
   SELECT a.empno, a.ename, b.dname
   FROM scott.emp a, jward.dept@hq.acme.com b
   WHERE a.deptno = b.deptno;
```

Figure 30–3 Views and Location Transparency



When users access this view, they do not need to know where the data is physically stored, or if data from more than one table is being accessed. Thus, it is easier for them to get required information. For example, the following query provides data from both the local and remote database table:

```
SELECT * FROM company;
```

The owner of the local view can grant only those object privileges on the local view that have been granted by the remote user. (The remote user is implied by the type

of database link). This is similar to privilege management for views that reference local data.

Using Synonyms to Create Location Transparency

Synonyms are useful in both distributed and nondistributed environments because they hide the identity of the underlying object, including its location in a distributed database system. If you must rename or move the underlying object, you only need to redefine the synonym; applications based on the synonym continue to function normally. Synonyms also simplify SQL statements for users in a distributed database system.

Creating Synonyms

You can create synonyms for the following:

- Tables
- Types
- Views
- Materialized views
- Sequences
- Procedures
- Functions
- Packages

All synonyms are schema objects that are stored in the data dictionary of the database in which they are created. To simplify remote table access through database links, a synonym can allow single-word access to remote data, hiding the specific object name and the location from users of the synonym.

The syntax to create a synonym is:

```
CREATE [PUBLIC] synonym_name
FOR [schema.]object_name[@database_link_name];
```

where:

- `PUBLIC` is a keyword specifying that this synonym is available to all users. Omitting this parameter makes a synonym private, and usable only by the creator. Public synonyms can be created only by a user with `CREATE PUBLIC SYNONYM` system privilege.

- *synonym_name* specifies the alternate object name to be referenced by users and applications.
- *schema* specifies the schema of the object specified in *object_name*. Omitting this parameter uses the schema of the creator as the schema of the object.
- *object_name* specifies either a table, view, sequence, materialized view, type, procedure, function or package as appropriate.
- *database_link_name* specifies the database link identifying the remote database and schema in which the object specified in *object_name* is located.

A synonym must be a uniquely named object for its schema. If a schema contains a schema object and a public synonym exists with the same name, then the database always finds the schema object when the user that owns the schema references that name.

Example: Creating a Public Synonym

Assume that in every database in a distributed database system, a public synonym is defined for the `scott.emp` table stored in the `hq` database:

```
CREATE PUBLIC SYNONYM emp FOR scott.emp@hq.acme.com;
```

You can design an employee management application without regard to where the application is used because the location of the table `scott.emp@hq.acme.com` is hidden by the public synonyms. SQL statements in the application access the table by referencing the public synonym `emp`.

Furthermore, if you move the `emp` table from the `hq` database to the `hr` database, then you only need to change the public synonyms on the nodes of the system. The employee management application continues to function properly on all nodes.

Managing Privileges and Synonyms

A synonym is a reference to an actual object. A user who has access to a synonym for a particular schema object must also have privileges on the underlying schema object itself. For example, if the user attempts to access a synonym but does not have privileges on the table it identifies, an error occurs indicating that the table or view does not exist.

Assume `scott` creates local synonym `emp` as an alias for remote object `scott.emp@sales.acme.com`. `scott` *cannot* grant object privileges on the synonym to another local user. `scott` cannot grant local privileges for the synonym because this operation amounts to granting privileges for the remote `emp` table on

the sales database, which is not allowed. This behavior is different from privilege management for synonyms that are aliases for local tables or views.

Therefore, you cannot manage local privileges when synonyms are used for location transparency. Security for the base object is controlled entirely at the remote node. For example, user admin cannot grant object privileges for the emp_syn synonym.

Unlike a database link referenced in a view or procedure definition, a database link referenced in a synonym is resolved by first looking for a private link owned by the schema in effect at the time the reference to the synonym is parsed. Therefore, to ensure the desired object resolution, it is especially important to specify the schema of the underlying object in the definition of a synonym.

Using Procedures to Create Location Transparency

PL/SQL program units called **procedures** can provide location transparency. You have these options:

- Using Local Procedures to Reference Remote Data
- Using Local Procedures to Call Remote Procedures
- Using Local Synonyms to Reference Remote Procedures

Using Local Procedures to Reference Remote Data

Procedures or functions (either standalone or in packages) can contain SQL statements that reference remote data. For example, consider the procedure created by the following statement:

```
CREATE PROCEDURE fire_emp (enum NUMBER) AS
BEGIN
  DELETE FROM emp@hq.acme.com
  WHERE empno = enum;
END;
```

When a user or application calls the fire_emp procedure, it is not apparent that a remote table is being modified.

A second layer of location transparency is possible when the statements in a procedure indirectly reference remote data using local procedures, views, or synonyms. For example, the following statement defines a local synonym:

```
CREATE SYNONYM emp FOR emp@hq.acme.com;
```

Given this synonym, you can create the fire_emp procedure using the following statement:

```
CREATE PROCEDURE fire_emp (enum NUMBER) AS
BEGIN
  DELETE FROM emp WHERE empno = enum;
END;
```

If you rename or move the table emp@hq, then you only need to modify the local synonym that references the table. None of the procedures and applications that call the procedure require modification.

Using Local Procedures to Call Remote Procedures

You can use a local procedure to call a remote procedure. The remote procedure can then execute the required DML. For example, assume that scott connects to local_db and creates the following procedure:

```
CONNECT scott/tiger@local_db

CREATE PROCEDURE fire_emp (enum NUMBER)
AS
BEGIN
  EXECUTE term_emp@hq.acme.com;
END;
```

Now, assume that scott connects to the remote database and creates the remote procedure:

```
CONNECT scott/tiger@hq.acme.com

CREATE PROCEDURE term_emp (enum NUMBER)
AS
BEGIN
  DELETE FROM emp WHERE empno = enum;
END;
```

When a user or application connected to local_db calls the fire_emp procedure, this procedure in turn calls the remote term_emp procedure on hq.acme.com.

Using Local Synonyms to Reference Remote Procedures

For example, scott connects to the local sales.acme.com database and creates the following procedure:

```
CREATE PROCEDURE fire_emp (enum NUMBER) AS
```

```
BEGIN
DELETE FROM emp@hq.acme.com
WHERE empno = enum;
END;
```

User peggy then connects to the supply.acme.com database and creates the following synonym for the procedure that scott created on the remote sales database:

```
SQL> CONNECT peggy/hill@supply
SQL> CREATE PUBLIC SYNONYM emp FOR scott.fire_emp@sales.acme.com;
```

A local user on supply can use this synonym to execute the procedure on sales.

Managing Procedures and Privileges

Assume a local procedure includes a statement that references a remote table or view. The owner of the local procedure can grant the execute privilege to any user, thereby giving that user the ability to execute the procedure and, indirectly, access remote data.

In general, procedures aid in security. Privileges for objects referenced within a procedure do not need to be explicitly granted to the calling users.

Managing Statement Transparency

The database allows the following standard DML statements to reference remote tables:

- SELECT (queries)
- INSERT
- UPDATE
- DELETE
- SELECT ... FOR UPDATE (not always supported in Heterogeneous Systems)
- LOCK TABLE

Queries including joins, aggregates, subqueries, and SELECT ... FOR UPDATE can reference any number of local and remote tables and views. For example, the following query joins information from two remote tables:

```
SELECT e.empno, e.ename, d.dname
  FROM scott.emp@sales.division3.acme.com e, jward.dept@hq.acme.com d
```

```
  WHERE e.deptno = d.deptno;
```

In a homogeneous environment, UPDATE, INSERT, DELETE, and LOCK TABLE statements can reference both local and remote tables. No programming is necessary to update remote data. For example, the following statement inserts new rows into the remote table emp in the scott.sales schema by selecting rows from the emp table in the jward schema in the local database:

```
INSERT INTO scott.emp@sales.division3.acme.com
  SELECT * FROM jward.emp;
```

Restrictions for Statement Transparency:

Several restrictions apply to statement transparency.

- Data manipulation language statements that update objects on a remote non-Oracle Database system cannot reference any objects on the local Oracle Database. For example, a statement such as the following will cause an error to be raised:

  ```
  INSERT INTO remote_table@link as SELECT * FROM local_table;
  ```

- Within a single SQL statement, all referenced LONG and LONG RAW columns, sequences, updated tables, and locked tables must be located at the same node.
- The database does not allow remote DDL statements (for example, CREATE, ALTER, and DROP) in homogeneous systems except through remote execution of procedures of the DBMS_SQL package, as in this example:

  ```
  DBMS_SQL.PARSE@link_name(crs, 'drop table emp', v7);
  ```

  Note that in Heterogeneous Systems, a pass-through facility lets you execute DDL.
- The LIST CHAINED ROWS clause of an ANALYZE statement cannot reference remote tables.
- In a distributed database system, the database always evaluates environmentally-dependent SQL functions such as SYSDATE, USER, UID, and USERENV with respect to the local server, no matter where the statement (or portion of a statement) executes.

> **Note:** Oracle Database supports the USERENV function for queries only.

- A number of performance restrictions relate to access of remote objects:
  - Remote views do not have statistical data.
  - Queries on partitioned tables may not be optimized.
  - No more than 20 indexes are considered for a remote table.
  - No more than 20 columns are used for a composite index.
- There is a restriction in the Oracle Database implementation of distributed read consistency that can cause one node to be in the past with respect to another node. In accordance with read consistency, a query may end up retrieving consistent, but out-of-date data. See "Managing Read Consistency" on page 33-25 to learn how to manage this problem.

  See Also: *PL/SQL Packages and Types Reference* for more information about the `DBMS_SQL` package

Managing a Distributed Database: Examples

This section presents examples illustrating management of database links:

- Example 1: Creating a Public Fixed User Database Link
- Example 2: Creating a Public Fixed User Shared Database Link
- Example 3: Creating a Public Connected User Database Link
- Example 4: Creating a Public Connected User Shared Database Link
- Example 5: Creating a Public Current User Database Link

Example 1: Creating a Public Fixed User Database Link

The following statements connect to the local database as `jane` and create a public fixed user database link to database `sales` for `scott`. The database is accessed through its net service name `sldb`:

```
CONNECT jane/doe@local

CREATE PUBLIC DATABASE LINK sales.division3.acme.com
  CONNECT TO scott IDENTIFIED BY tiger
  USING 'sldb';
```

After executing these statements, any user connected to the local database can use the sales.division3.acme.com database link to connect to the remote database. Each user connects to the schema scott in the remote database.

To access the table emp table in scott's remote schema, a user can issue the following SQL query:

```
SELECT * FROM emp@sales.division3.acme.com;
```

Note that each application or user session creates a separate connection to the common account on the server. The connection to the remote database remains open for the duration of the application or user session.

Example 2: Creating a Public Fixed User Shared Database Link

The following example connects to the local database as dana and creates a public link to the sales database (using its net service name sldb). The link allows a connection to the remote database as scott and authenticates this user as scott:

```
CONNECT dana/sculley@local

CREATE SHARED PUBLIC DATABASE LINK sales.division3.acme.com
  CONNECT TO scott IDENTIFIED BY tiger
  AUTHENTICATED BY scott IDENTIFIED BY tiger
  USING 'sldb';
```

Now, any user connected to the local shared server can use this database link to connect to the remote sales database through a shared server process. The user can then query tables in the scott schema.

In the preceding example, each local shared server can establish one connection to the remote server. Whenever a local shared server process needs to access the remote server through the sales.division3.acme.com database link, the local shared server process reuses established network connections.

Example 3: Creating a Public Connected User Database Link

The following example connects to the local database as larry and creates a public link to the database with the net service name sldb:

```
CONNECT larry/oracle@local

CREATE PUBLIC DATABASE LINK redwood
  USING 'sldb';
```

Any user connected to the local database can use the redwood database link. The connected user in the local database who uses the database link determines the remote schema.

If scott is the connected user and uses the database link, then the database link connects to the remote schema scott. If fox is the connected user and uses the database link, then the database link connects to remote schema fox.

The following statement fails for local user fox in the local database when the remote schema fox cannot resolve the emp schema object. That is, if the fox schema in the sales.division3.acme.com does not have emp as a table, view, or (public) synonym, an error will be returned.

```
CONNECT fox/mulder@local

SELECT * FROM emp@redwood;
```

Example 4: Creating a Public Connected User Shared Database Link

The following example connects to the local database as neil and creates a shared, public link to the sales database (using its net service name sldb). The user is authenticated by the userid / password of crazy/horse. The following statement creates a public, connected user, shared database link:

```
CONNECT neil/young@local

CREATE SHARED PUBLIC DATABASE LINK sales.division3.acme.com
  AUTHENTICATED BY crazy IDENTIFIED BY horse
  USING 'sldb';
```

Each user connected to the local server can use this shared database link to connect to the remote database and query the tables in the corresponding remote schema.

Each local, shared server process establishes one connection to the remote server. Whenever a local server process needs to access the remote server through the sales.division3.acme.com database link, the local process reuses established network connections, even if the connected user is a different user.

If this database link is used frequently, eventually every shared server in the local database will have a remote connection. At this point, no more physical connections are needed to the remote server, even if new users use this shared database link.

Example 5: Creating a Public Current User Database Link

The following example connects to the local database as the connected user and creates a public link to the `sales` database (using its net service name `sldb`). The following statement creates a public current user database link:

```
CONNECT bart/simpson@local

CREATE PUBLIC DATABASE LINK sales.division3.acme.com
  CONNECT TO CURRENT_USER
  USING 'sldb';
```

> **Note:** To use this link, the current user must be a global user.

The consequences of this database link are as follows:

Assume `scott` creates local procedure `fire_emp` that deletes a row from the remote `emp` table, and grants execute privilege on `fire_emp` to `ford`.

```
CONNECT scott/tiger@local_db

CREATE PROCEDURE fire_emp (enum NUMBER)
AS
BEGIN
   DELETE FROM emp@sales.division3.acme.com
   WHERE empno=enum;
END;

GRANT EXECUTE ON fire_emp TO ford;
```

Now, assume that `ford` connects to the local database and runs `scott`'s procedure:

```
CONNECT ford/fairlane@local_db

EXECUTE PROCEDURE scott.fire_emp (enum 10345);
```

When `ford` executes the procedure `scott.fire_emp`, the procedure runs under `scott`'s privileges. Because a current user database link is used, the connection is established to `scott`'s remote schema, not `ford`'s remote schema. Note that `scott` must be a global user while `ford` does not have to be a global user.

Note: If a connected user database link were used instead, the connection would be to `ford`'s remote schema. For more information about invoker rights and privileges, see the *PL/SQL User's Guide and Reference*.

You can accomplish the same result by using a fixed user database link to `scott`'s remote schema. With fixed user database links, however, security can be compromised because `scott`'s username and password are available in readable format in the database.

31

Developing Applications for a Distributed Database System

This chapter describes considerations important when developing an application to run in a distributed database system. It contains the following topics:

- Managing the Distribution of Application Data
- Controlling Connections Established by Database Links
- Maintaining Referential Integrity in a Distributed System
- Tuning Distributed Queries
- Handling Errors in Remote Procedures

> **See Also:** *Oracle Database Application Developer's Guide - Fundamentals* for more information about application development in an Oracle Database environment

Managing the Distribution of Application Data

In a distributed database environment, coordinate with the database administrator to determine the best location for the data. Some issues to consider are:

- Number of transactions posted from each location
- Amount of data (portion of table) used by each node
- Performance characteristics and reliability of the network
- Speed of various nodes, capacities of disks
- Importance of a node or link when it is unavailable
- Need for referential integrity among tables

Controlling Connections Established by Database Links

When a global object name is referenced in a SQL statement or remote procedure call, database links establish a connection to a session in the remote database on behalf of the local user. The remote connection and session are only created if the connection has not already been established previously for the local user session.

The connections and sessions established to remote databases persist for the duration of the local user's session, unless the application or user explicitly terminates them. Note that when you issue a SELECT statement across a database link, a transaction lock is placed on the undo segments. To rerelease the segment, you must issue a COMMIT or ROLLBACK statement.

Terminating remote connections established using database links is useful for disconnecting high cost connections that are no longer required by the application. You can terminate a remote connection and session using the ALTER SESSION statement with the CLOSE DATABASE LINK clause. For example, assume you issue the following transactions:

```
SELECT * FROM emp@sales;
COMMIT;
```

The following statement terminates the session in the remote database pointed to by the sales database link:

```
ALTER SESSION CLOSE DATABASE LINK sales;
```

To close a database link connection in your user session, you must have the ALTER SESSION system privilege.

Note: Before closing a database link, first close all cursors that use the link and then end your current transaction if it uses the link.

See Also: *Oracle Database SQL Reference* for more information about the ALTER SESSION statement

Maintaining Referential Integrity in a Distributed System

If a part of a distributed statement fails, for example, due to an integrity constraint violation, the database returns error number ORA-02055. Subsequent statements or procedure calls return error number ORA-02067 until a ROLLBACK or ROLLBACK TO SAVEPOINT is issued.

Design your application to check for any returned error messages that indicate that a portion of the distributed update has failed. If you detect a failure, you should roll back the entire transaction before allowing the application to proceed.

The database does not permit declarative referential integrity constraints to be defined across nodes of a distributed system. In other words, a declarative referential integrity constraint on one table cannot specify a foreign key that references a primary or unique key of a remote table. Nevertheless, you can maintain parent/child table relationships across nodes using triggers.

If you decide to define referential integrity across the nodes of a distributed database using triggers, be aware that network failures can limit the accessibility of not only the parent table, but also the child table. For example, assume that the child table is in the `sales` database and the parent table is in the `hq` database. If the network connection between the two databases fails, some DML statements against the child table (those that insert rows into the child table or update a foreign key value in the child table) cannot proceed because the referential integrity triggers must have access to the parent table in the `hq` database.

> **See Also:** *Oracle Database Application Developer's Guide - Fundamentals* for more information about using triggers to enforce referential integrity

Tuning Distributed Queries

The local Oracle Database server breaks the distributed query into a corresponding number of remote queries, which it then sends to the remote nodes for execution. The remote nodes execute the queries and send the results back to the local node. The local node then performs any necessary post-processing and returns the results to the user or application.

You have several options for designing your application to optimize query processing. This section contains the following topics:

- Using Collocated Inline Views
- Using Cost-Based Optimization
- Using Hints
- Analyzing the Execution Plan

Using Collocated Inline Views

The most effective way of optimizing distributed queries is to access the remote databases as little as possible and to retrieve only the required data.

For example, assume you reference five remote tables from two different remote databases in a distributed query and have a complex filter (for example, `WHERE r1.salary + r2.salary > 50000`). You can improve the performance of the query by rewriting the query to access the remote databases once and to apply the filter at the remote site. This rewrite causes less data to be transferred to the query execution site.

Rewriting your query to access the remote database once is achieved by using collocated inline views. The following terms need to be defined:

- **Collocated**

  Two or more tables located in the same database.

- **Inline view**

  A SELECT statement that is substituted for a table in a parent SELECT statement. The embedded SELECT statement, shown within the parentheses is an example of an inline view:

  ```
  SELECT e.empno,e.ename,d.deptno,d.dname
     FROM (SELECT empno, ename from
           emp@orc1.world) e, dept d;
  ```

- **Collocated inline view**

  An inline view that selects data from multiple tables from a single database only. It reduces the amount of times that the remote database is accessed, improving the performance of a distributed query.

Oracle recommends that you form your distributed query using collocated inline views to increase the performance of your distributed query. Oracle Database cost-based optimization can transparently rewrite many of your distributed queries to take advantage of the performance gains offered by collocated inline views.

Using Cost-Based Optimization

In addition to rewriting your queries with collocated inline views, the cost-based optimization method optimizes distributed queries according to the gathered statistics of the referenced tables and the computations performed by the optimizer.

For example, cost-based optimization analyzes the following query. The example assumes that table statistics are available. Note that it analyzes the query inside a CREATE TABLE statement:

```
CREATE TABLE AS (
                 SELECT l.a, l.b, r1.c, r1.d, r1.e, r2.b, r2.c
                 FROM local l, remote1 r1, remote2 r2
                    WHERE l.c = r.c
                    AND r1.c = r2.c
                    AND r.e > 300
                );
```

and rewrites it as:

```
CREATE TABLE AS (
                 SELECT l.a, l.b, v.c, v.d, v.e
                 FROM (
                        SELECT r1.c, r1.d, r1.e, r2.b, r2.c
                         FROM remote1 r1, remote2 r2
                         WHERE r1.c = r2.c
                         AND r1.e > 300
                      ) v, local l
                 WHERE l.c = r1.c
                );
```

The alias v is assigned to the inline view, which can then be referenced as a table in the preceding SELECT statement. Creating a collocated inline view reduces the amount of queries performed at a remote site, thereby reducing costly network traffic.

How Does Cost-Based Optimization Work?

The main task of optimization is to rewrite a distributed query to use collocated inline views. This optimization is performed in three steps:

1. All mergeable views are merged.
2. Optimizer performs collocated query block test.
3. Optimizer rewrites query using collocated inline views.

After the query is rewritten, it is executed and the data set is returned to the user.

While cost-based optimization is performed transparently to the user, it is unable to improve the performance of several distributed query scenarios. Specifically, if your distributed query contains any of the following, cost-based optimization is not effective:

- Aggregates
- Subqueries
- Complex SQL

If your distributed query contains one of these elements, see "Using Hints" on page 31-7 to learn how you can modify your query and use hints to improve the performance of your distributed query.

Setting Up Cost-Based Optimization

After you have set up your system to use cost-based optimization to improve the performance of distributed queries, the operation is transparent to the user. In other words, the optimization occurs automatically when the query is issued.

You need to complete the following tasks to set up your system to take advantage of cost-based optimization:

- Setting Up the Environment
- Analyzing Tables

Setting Up the Environment To enable cost-based optimization, set the `OPTIMIZER_MODE` initialization parameter to `CHOOSE` or `COST`. You can set this parameter by:

- Modifying the `OPTIMIZER_MODE` parameter in the initialization parameter file
- Setting it at session level by issuing an `ALTER SESSION` statement

Issue one of the following statements to set the `OPTIMIZER_MODE` initialization parameter at the session level:

```
ALTER SESSION OPTIMIZER_MODE = CHOOSE;
ALTER SESSION OPTIMIZER_MODE = COST;
```

> **See Also:** *Oracle Database Performance Tuning Guide* for information on setting the `OPTIMIZER_MODE` initialization parameter in the parameter file and for configuring your system to use a cost-based optimization method

Analyzing Tables For cost-based optimization to select the most efficient path for a distributed query, you must provide accurate statistics for the tables involved. You do this using the `DBMS_STATS` package or the `ANALYZE` statement.

Note: You must connect locally with respect to the tables when executing the DBMS_STATS procedure or ANALYZE statement. For example, you cannot execute the following:

```
ANALYZE TABLE remote@remote.com COMPUTE STATISTICS;
```

You must first connect to the remote site and then execute the preceding ANALYZE statement, or the equivalent DBMS_STATS procedure.

The following DBMS_STATS procedures enable the gathering of certain classes of optimizer statistics:

- GATHER_INDEX_STATS
- GATHER_TABLE_STATS
- GATHER_SCHEMA_STATS
- GATHER_DATABASE_STATS

For example, assume that distributed transactions routinely access the `scott.dept` table. To ensure that the cost-based optimizer is still picking the best plan, execute the following:

```
BEGIN
   DBMS_STATS.GATHER_TABLE_STATS ('scott', 'dept');
END;
```

See Also:

- *Oracle Database Performance Tuning Guide* for information about generating statistics
- *PL/SQL Packages and Types Reference* for additional information on using the DBMS_STATS package

Using Hints

If a statement is not sufficiently optimized, then you can use hints to extend the capability of cost-based optimization. Specifically, if you write your own query to utilize collocated inline views, instruct the cost-based optimizer not to rewrite your distributed query.

Additionally, if you have special knowledge about the database environment (such as statistics, load, network and CPU limitations, distributed queries, and so forth),

you can specify a hint to guide cost-based optimization. For example, if you have written your own optimized query using collocated inline views that are based on your knowledge of the database environment, specify the NO_MERGE hint to prevent the optimizer from rewriting your query.

This technique is especially helpful if your distributed query contains an aggregate, subquery, or complex SQL. Because this type of distributed query cannot be rewritten by the optimizer, specifying NO_MERGE causes the optimizer to skip the steps described in "How Does Cost-Based Optimization Work?" on page 31-5.

The DRIVING_SITE hint lets you define a remote site to act as the query execution site. In this way, the query executes on the remote site, which then returns the data to the local site. This hint is especially helpful when the remote site contains the majority of the data.

> **See Also:** *Oracle Database Performance Tuning Guide* for more information about using hints

Using the NO_MERGE Hint

The NO_MERGE hint prevents the database from merging an inline view into a potentially noncollocated SQL statement (see "Using Hints" on page 31-7). This hint is embedded in the SELECT statement and can appear either at the beginning of the SELECT statement with the inline view as an argument or in the query block that defines the inline view.

```
/* with argument */

SELECT /*+NO_MERGE(v)*/ t1.x, v.avg_y
   FROM t1, (SELECT x, AVG(y) AS avg_y FROM t2 GROUP BY x) v,
   WHERE t1.x = v.x AND t1.y = 1;

/* in query block */

SELECT t1.x, v.avg_y
   FROM t1, (SELECT /*+NO_MERGE*/ x, AVG(y) AS avg_y FROM t2 GROUP BY x) v,
   WHERE t1.x = v.x AND t1.y = 1;
```

Typically, you use this hint when you have developed an optimized query based on your knowledge of your database environment.

Using the DRIVING_SITE Hint

The DRIVING_SITE hint lets you specify the site where the query execution is performed. It is best to let cost-based optimization determine where the execution

should be performed, but if you prefer to override the optimizer, you can specify the execution site manually.

Following is an example of a SELECT statement with a DRIVING_SITE hint:

```
SELECT /*+DRIVING_SITE(dept)*/ * FROM emp, dept@remote.com
   WHERE emp.deptno = dept.deptno;
```

Analyzing the Execution Plan

An important aspect to tuning distributed queries is analyzing the execution plan. The feedback that you receive from your analysis is an important element to testing and verifying your database. Verification becomes especially important when you want to compare plans. For example, comparing the execution plan for a distributed query optimized by cost-based optimization to a plan for a query manually optimized using hints, collocated inline views, and other techniques.

> **See Also:** *Oracle Database Performance Tuning Guide* for detailed information about execution plans, the EXPLAIN PLAN statement, and how to interpret the results

Preparing the Database to Store the Plan

Before you can view the execution plan for the distributed query, prepare the database to store the execution plan. You can perform this preparation by executing a script. Execute the following script to prepare your database to store an execution plan:

```
SQL> @UTLXPLAN.SQL
```

> **Note:** The utlxplan.sql file can be found in the $ORACLE_HOME/rdbms/admin directory.

After you execute utlxplan.sql, a table, PLAN_TABLE, is created in the current schema to temporarily store the execution plan.

Generating the Execution Plan

After you have prepared the database to store the execution plan, you are ready to view the plan for a specified query. Instead of directly executing a SQL statement, append the statement to the EXPLAIN PLAN FOR clause. For example, you can execute the following:

```
EXPLAIN PLAN FOR
   SELECT d.dname
   FROM dept d
      WHERE d.deptno
      IN (SELECT deptno
          FROM emp@orc2.world
          GROUP BY deptno
          HAVING COUNT (deptno) >3
          )
/
```

Viewing the Execution Plan

After you have executed the preceding SQL statement, the execution plan is stored temporarily in the PLAN_TABLE that you created earlier. To view the results of the execution plan, execute the following script:

```
@UTLXPLS.SQL
```

> **Note:** The `utlxpls.sql` can be found in the $ORACLE_HOME/rdbms/admin directory.

Executing the `utlxpls.sql` script displays the execution plan for the SELECT statement that you specified. The results are formatted as follows:

```
Plan Table
--------------------------------------------------------------------------------
| Operation                 |  Name    |  Rows | Bytes|  Cost  | Pstart| Pstop |
--------------------------------------------------------------------------------
| SELECT STATEMENT          |          |       |      |        |       |       |
|  NESTED LOOPS             |          |       |      |        |       |       |
|   VIEW                    |          |       |      |        |       |       |
|    REMOTE                 |          |       |      |        |       |       |
|   TABLE ACCESS BY INDEX RO|DEPT      |       |      |        |       |       |
|    INDEX UNIQUE SCAN      |PK_DEPT   |       |      |        |       |       |
--------------------------------------------------------------------------------
```

If you are manually optimizing distributed queries by writing your own collocated inline views or using hints, it is best to generate an execution plan before and after your manual optimization. With both execution plans, you can compare the effectiveness of your manual optimization and make changes as necessary to improve the performance of the distributed query.

To view the SQL statement that will be executed at the remote site, execute the following select statement:

```
SELECT OTHER
FROM PLAN_TABLE
   WHERE operation = 'REMOTE';
```

Following is sample output:

```
SELECT DISTINCT "A1"."DEPTNO" FROM "EMP" "A1"
   GROUP BY "A1"."DEPTNO" HAVING COUNT("A1"."DEPTNO")>3
```

Note: If you are having difficulty viewing the entire contents of the OTHER column, execute the following SQL*Plus command:

```
SET LONG 9999999
```

Handling Errors in Remote Procedures

When the database executes a procedure locally or at a remote location, four types of exceptions can occur:

- PL/SQL user-defined exceptions, which must be declared using the keyword EXCEPTION
- PL/SQL predefined exceptions such as the NO_DATA_FOUND keyword
- SQL errors such as ORA-00900 and ORA-02015
- Application exceptions generated using the RAISE_APPLICATION_ERROR() procedure

When using local procedures, you can trap these messages by writing an exception handler such as the following

```
BEGIN
  ...
EXCEPTION
   WHEN ZERO_DIVIDE THEN
   /* ... handle the exception */
END;
```

Notice that the WHEN clause requires an exception name. If the exception does not have a name, for example, exceptions generated with RAISE_APPLICATION_ERROR, you can assign one using PRAGMA_EXCEPTION_INIT. For example:

```
DECLARE
  null_salary EXCEPTION;
  PRAGMA EXCEPTION_INIT(null_salary, -20101);
BEGIN
  ...
  RAISE_APPLICATION_ERROR(-20101, 'salary is missing');
...
EXCEPTION
  WHEN null_salary THEN
  ...
END;
```

When calling a remote procedure, exceptions can be handled by an exception handler in the local procedure. The remote procedure must return an error number to the local, calling procedure, which then handles the exception as shown in the previous example. Note that PL/SQL user-defined exceptions always return `ORA-06510` to the local procedure.

Therefore, it is not possible to distinguish between two different user-defined exceptions based on the error number. All other remote exceptions can be handled in the same manner as local exceptions.

> **See Also:** *PL/SQL User's Guide and Reference* for more information about PL/SQL procedures

32

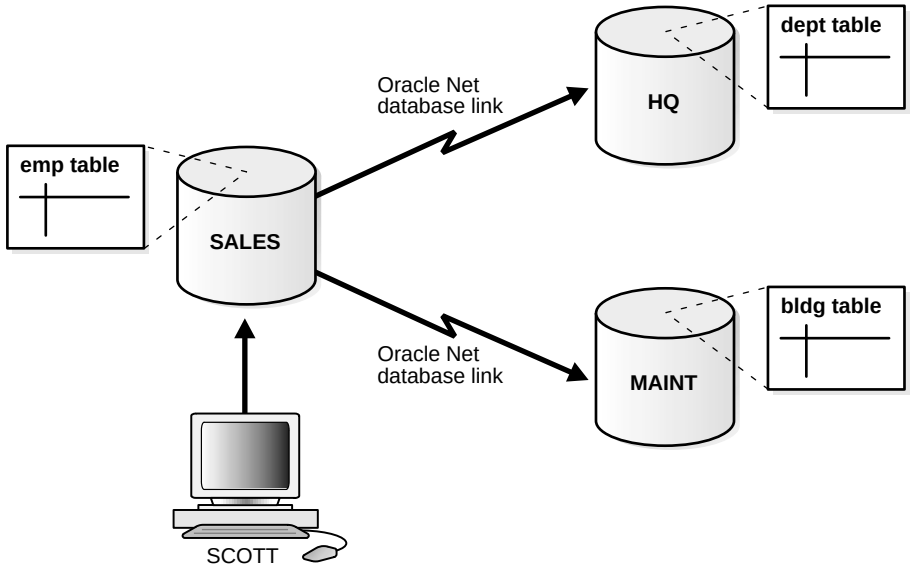
Distributed Transactions Concepts

This chapter describes what distributed transactions are and how Oracle Database maintains their integrity. The following topics are contained in this chapter:

What Are Distributed Transactions?

A **distributed transaction** includes one or more statements that, individually or as a group, update data on two or more distinct nodes of a distributed database. For example, assume the database configuration depicted in Figure 32–1:

Figure 32–1 Distributed System



The following distributed transaction executed by `scott` updates the local `sales` database, the remote `hq` database, and the remote `maint` database:

```
UPDATE scott.dept@hq.us.acme.com
  SET loc = 'REDWOOD SHORES'
  WHERE deptno = 10;
UPDATE scott.emp
  SET deptno = 11
  WHERE deptno = 10;
UPDATE scott.bldg@maint.us.acme.com
  SET room = 1225
  WHERE room = 1163;
COMMIT;
```

> **Note:** If all statements of a transaction reference only a single remote node, then the transaction is remote, not distributed.

There are two types of permissible operations in distributed transactions:

- DML and DDL Transactions
- Transaction Control Statements

DML and DDL Transactions

The following are the DML and DDL operations supported in a distributed transaction:

- CREATE TABLE AS SELECT
- DELETE
- INSERT (default and direct load)
- LOCK TABLE
- SELECT
- SELECT FOR UPDATE

You can execute DML and DDL statements in parallel, and INSERT direct load statements serially, but note the following restrictions:

- All remote operations must be SELECT statements.
- These statements must not be clauses in another distributed transaction.
- If the table referenced in the *table_expression_clause* of an INSERT, UPDATE, or DELETE statement is remote, then execution is serial rather than parallel.
- You cannot perform remote operations after issuing parallel DML / DDL or direct load INSERT.
- If the transaction begins using XA or OCI, it executes serially.
- No loopback operations can be performed on the transaction originating the parallel operation. For example, you cannot reference a remote object that is actually a synonym for a local object.
- If you perform a distributed operation other than a SELECT in the transaction, no DML is parallelized.

Transaction Control Statements

The following are the supported transaction control statements:

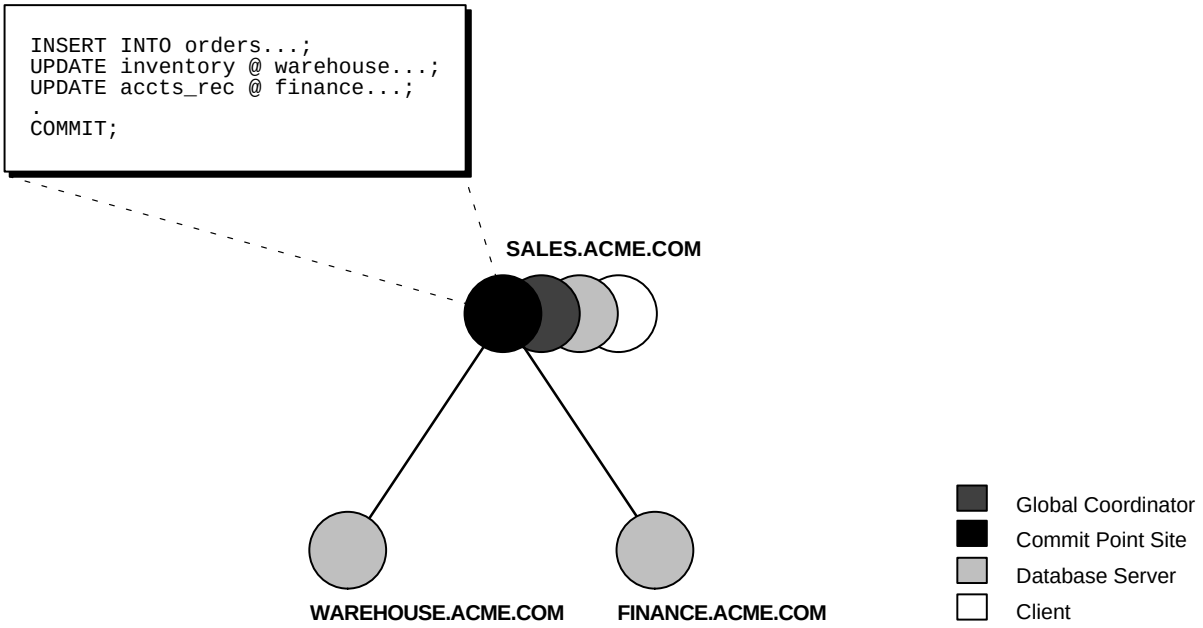
- COMMIT
- ROLLBACK
- SAVEPOINT

See Also: *Oracle Database SQL Reference* for more information about these SQL statements

Session Trees for Distributed Transactions

As the statements in a distributed transaction are issued, the database defines a **session tree** of all nodes participating in the transaction. A session tree is a hierarchical model that describes the relationships among sessions and their roles. Figure 32–2 illustrates a session tree:

Figure 32–2 Example of a Session Tree



All nodes participating in the session tree of a distributed transaction assume one or more of the following roles:

Role	Description
Client	A node that references information in a database belonging to a different node.

Role	Description
Database server	A node that receives a request for information from another node.
Global coordinator	The node that originates the distributed transaction.
Local coordinator	A node that is forced to reference data on other nodes to complete its part of the transaction.
Commit point site	The node that commits or rolls back the transaction as instructed by the global coordinator.

The role a node plays in a distributed transaction is determined by:

- Whether the transaction is local or remote
- The **commit point strength** of the node ("Commit Point Site" on page 32-6)
- Whether all requested data is available at a node, or whether other nodes need to be referenced to complete the transaction
- Whether the node is read-only

Clients

A node acts as a client when it references information from a database on another node. The referenced node is a database server. In Figure 32–2, the node `sales` is a client of the nodes that host the `warehouse` and `finance` databases.

Database Servers

A database server is a node that hosts a database from which a client requests data.

In Figure 32–2, an application at the `sales` node initiates a distributed transaction that accesses data from the `warehouse` and `finance` nodes. Therefore, `sales.acme.com` has the role of client node, and `warehouse` and `finance` are both database servers. In this example, `sales` is a database server *and* a client because the application also modifies data in the `sales` database.

Local Coordinators

A node that must reference data on other nodes to complete its part in the distributed transaction is called a local coordinator. In Figure 32–2, `sales` is a local coordinator because it coordinates the nodes it directly references: `warehouse` and

finance. The node sales also happens to be the global coordinator because it coordinates all the nodes involved in the transaction.

A local coordinator is responsible for coordinating the transaction among the nodes it communicates directly with by:

- Receiving and relaying transaction status information to and from those nodes
- Passing queries to those nodes
- Receiving queries from those nodes and passing them on to other nodes
- Returning the results of queries to the nodes that initiated them

Global Coordinator

The node where the distributed transaction originates is called the global coordinator. The database application issuing the distributed transaction is directly connected to the node acting as the global coordinator. For example, in Figure 32–2, the transaction issued at the node sales references information from the database servers warehouse and finance. Therefore, sales.acme.com is the global coordinator of this distributed transaction.

The global coordinator becomes the parent or root of the session tree. The global coordinator performs the following operations during a distributed transaction:

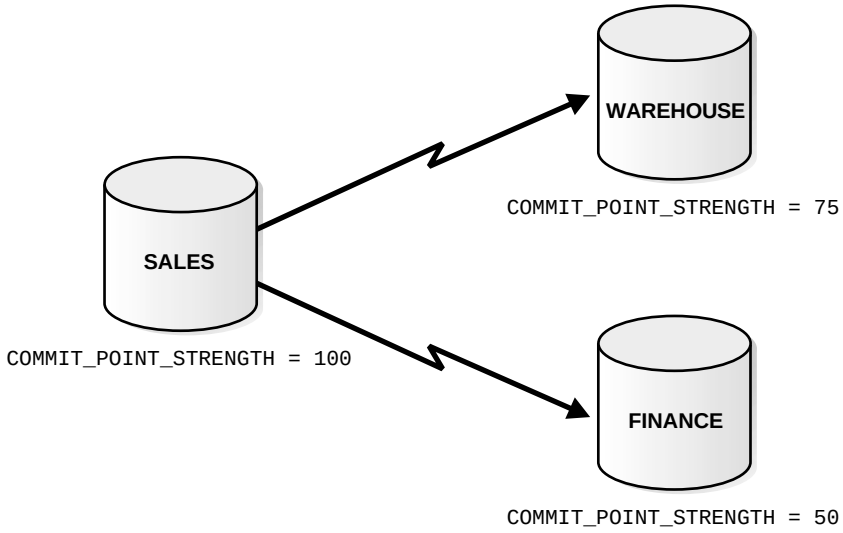
- Sends all of the distributed transaction SQL statements, remote procedure calls, and so forth to the directly referenced nodes, thus forming the session tree
- Instructs all directly referenced nodes other than the commit point site to prepare the transaction
- Instructs the commit point site to initiate the global commit of the transaction if all nodes prepare successfully
- Instructs all nodes to initiate a global rollback of the transaction if there is an abort response

Commit Point Site

The job of the commit point site is to initiate a commit or roll back operation as instructed by the global coordinator. The system administrator always designates one node to be the commit point site in the session tree by assigning all nodes a commit point strength. The node selected as commit point site should be the node that stores the most critical data.

Figure 32–3 illustrates an example of distributed system, with `sales` serving as the commit point site:

Figure 32–3 Commit Point Site



The commit point site is distinct from all other nodes involved in a distributed transaction in these ways:

- The commit point site never enters the prepared state. Consequently, if the commit point site stores the most critical data, this data never remains in-doubt, even if a failure occurs. In failure situations, failed nodes remain in a prepared state, holding necessary locks on data until in-doubt transactions are resolved.
- The commit point site commits before the other nodes involved in the transaction. In effect, the outcome of a distributed transaction at the commit point site determines whether the transaction at all nodes is committed or rolled back: the other nodes follow the lead of the commit point site. The global coordinator ensures that all nodes complete the transaction in the same manner as the commit point site.

How a Distributed Transaction Commits

A distributed transaction is considered committed after all non-commit-point sites are prepared, and the transaction has been actually committed at the commit point

site. The redo log at the commit point site is updated as soon as the distributed transaction is committed at this node.

Because the commit point log contains a record of the commit, the transaction is considered committed even though some participating nodes may still be only in the prepared state and the transaction not yet actually committed at these nodes. In the same way, a distributed transaction is considered *not* committed if the commit has not been logged at the commit point site.

Commit Point Strength

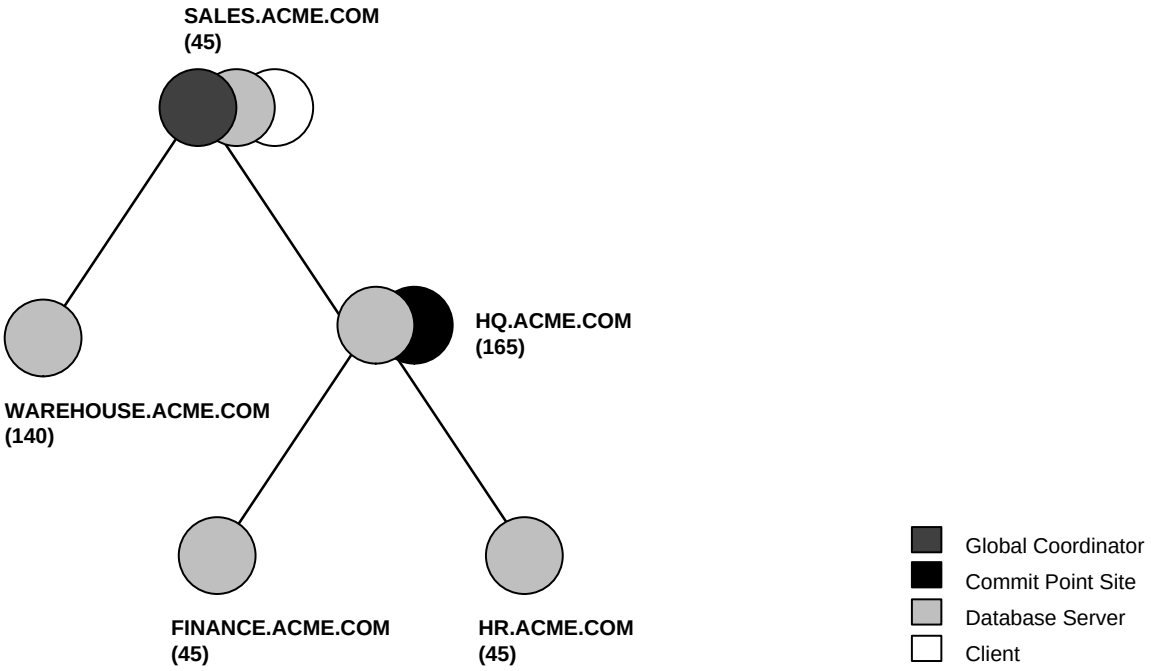
Every database server must be assigned a commit point strength. If a database server is referenced in a distributed transaction, the value of its commit point strength determines which role it plays in the two-phase commit. Specifically, the commit point strength determines whether a given node is the commit point site in the distributed transaction and thus commits before all of the other nodes. This value is specified using the initialization parameter `COMMIT_POINT_STRENGTH`. This section explains how the database determines the commit point site.

The commit point site, which is determined at the beginning of the prepare phase, is selected only from the nodes participating in the transaction. The following sequence of events occurs:

1. Of the nodes directly referenced by the global coordinator, the database selects the node with the highest commit point strength as the commit point site.
2. The initially-selected node determines if any of the nodes from which it has to obtain information for this transaction has a higher commit point strength.
3. Either the node with the highest commit point strength directly referenced in the transaction or one of its servers with a higher commit point strength becomes the commit point site.
4. After the final commit point site has been determined, the global coordinator sends prepare responses to all nodes participating in the transaction.

Figure 32–4 shows in a sample session tree the commit point strengths of each node (in parentheses) and shows the node chosen as the commit point site:

Figure 32–4 Commit Point Strengths and Determination of the Commit Point Site



The following conditions apply when determining the commit point site:

- A read-only node cannot be the commit point site.
- If multiple nodes directly referenced by the global coordinator have the same commit point strength, then the database designates one of these as the commit point site.
- If a distributed transaction ends with a rollback, then the prepare and commit phases are not needed. Consequently, the database never determines a commit point site. Instead, the global coordinator sends a ROLLBACK statement to all nodes and ends the processing of the distributed transaction.

As Figure 32–4 illustrates, the commit point site and the global coordinator can be different nodes of the session tree. The commit point strength of each node is communicated to the coordinators when the initial connections are made. The coordinators retain the commit point strengths of each node they are in direct communication with so that commit point sites can be efficiently selected during two-phase commits. Therefore, it is not necessary for the commit point strength to be exchanged between a coordinator and a node each time a commit occurs.

See Also:

- "Specifying the Commit Point Strength of a Node" on page 33-1 to learn how to set the commit point strength of a node
- *Oracle Database Reference* for more information about the initialization parameter `COMMIT_POINT_STRENGTH`

Two-Phase Commit Mechanism

Unlike a transaction on a local database, a distributed transaction involves altering data on multiple databases. Consequently, distributed transaction processing is more complicated, because the database must coordinate the committing or rolling back of the changes in a transaction as a self-contained unit. In other words, the entire transaction commits, or the entire transaction rolls back.

The database ensures the integrity of data in a distributed transaction using the **two-phase commit mechanism**. In the **prepare phase**, the initiating node in the transaction asks the other participating nodes to promise to commit or roll back the transaction. During the **commit phase**, the initiating node asks all participating nodes to commit the transaction. If this outcome is not possible, then all nodes are asked to roll back.

All participating nodes in a distributed transaction should perform the same action: they should either all commit or all perform a rollback of the transaction. The database automatically controls and monitors the commit or rollback of a distributed transaction and maintains the integrity of the **global database** (the collection of databases participating in the transaction) using the two-phase commit mechanism. This mechanism is completely transparent, requiring no programming on the part of the user or application developer.

The commit mechanism has the following distinct phases, which the database performs automatically whenever a user commits a distributed transaction:

Phase	Description
Prepare phase	The initiating node, called the **global coordinator**, asks participating nodes other than the commit point site to promise to commit or roll back the transaction, even if there is a failure. If any node cannot prepare, the transaction is rolled back.
Commit phase	If all participants respond to the coordinator that they are prepared, then the coordinator asks the commit point site to commit. After it commits, the coordinator asks all other nodes to commit the transaction.

Phase	Description
Forget phase	The global coordinator forgets about the transaction.

This section contains the following topics:

- Prepare Phase
- Commit Phase
- Forget Phase

Prepare Phase

The first phase in committing a distributed transaction is the prepare phase. In this phase, the database does not actually commit or roll back the transaction. Instead, all nodes referenced in a distributed transaction (except the commit point site, described in the "Commit Point Site" on page 32-6) are told to prepare to commit. By preparing, a node:

- Records information in the redo logs so that it can subsequently either commit or roll back the transaction, regardless of intervening failures
- Places a distributed lock on modified tables, which prevents reads

When a node responds to the global coordinator that it is prepared to commit, the prepared node *promises* to either commit or roll back the transaction later, but does not make a unilateral decision on whether to commit or roll back the transaction. The promise means that if an instance failure occurs at this point, the node can use the redo records in the online log to recover the database back to the prepare phase.

> **Note:** Queries that start after a node has prepared cannot access the associated locked data until all phases complete. The time is insignificant unless a failure occurs (see "Deciding How to Handle In-Doubt Transactions" on page 33-7).

Types of Responses in the Prepare Phase

When a node is told to prepare, it can respond in the following ways:

Response	Meaning
Prepared	Data on the node has been modified by a statement in the distributed transaction, and the node has successfully prepared.
Read-only	No data on the node has been, or can be, modified (only queried), so no preparation is necessary.
Abort	The node cannot successfully prepare.

Prepared Response When a node has successfully prepared, it issues a **prepared message**. The message indicates that the node has records of the changes in the online log, so it is prepared either to commit or perform a rollback. The message also guarantees that locks held for the transaction can survive a failure.

Read-Only Response When a node is asked to prepare, and the SQL statements affecting the database do not change any data on the node, the node responds with a **read-only message**. The message indicates that the node will not participate in the commit phase.

There are three cases in which all or part of a distributed transaction is read-only:

Case	Conditions	Consequence
Partially read-only	Any of the following occurs: ■ Only queries are issued at one or more nodes. ■ No data is changed. ■ Changes rolled back due to triggers firing or constraint violations.	The read-only nodes recognize their status when asked to prepare. They give their local coordinators a read-only response. Thus, the commit phase completes faster because the database eliminates read-only nodes from subsequent processing.
Completely read-only with prepare phase	All of following occur: ■ No data changes. ■ Transaction is *not* started with `SET TRANSACTION READ ONLY` statement.	All nodes recognize that they are read-only during prepare phase, so no commit phase is required. The global coordinator, not knowing whether all nodes are read-only, must still perform the prepare phase.

Case	Conditions	Consequence
Completely read-only without two-phase commit	All of following occur: ■ No data changes. ■ Transaction *is* started with `SET TRANSACTION READ ONLY` statement.	Only queries are allowed in the transaction, so global coordinator does not have to perform two-phase commit. Changes by other transactions do not degrade global transaction-level read consistency because of global SCN coordination among nodes. The transaction does not use undo segments.

Note that if a distributed transaction is set to read-only, then it does not use undo segments. If many users connect to the database and their transactions are *not* set to `READ ONLY`, then they allocate undo space even if they are only performing queries.

Abort Response When a node cannot successfully prepare, it performs the following actions:

1. Releases resources currently held by the transaction and rolls back the local portion of the transaction.
2. Responds to the node that referenced it in the distributed transaction with an abort message.

These actions then propagate to the other nodes involved in the distributed transaction so that they can roll back the transaction and guarantee the integrity of the data in the global database. This response enforces the primary rule of a distributed transaction: *all nodes involved in the transaction either all commit or all roll back the transaction at the same logical time.*

Steps in the Prepare Phase

To complete the prepare phase, each node excluding the commit point site performs the following steps:

1. The node requests that its **descendants**, that is, the nodes subsequently referenced, prepare to commit.
2. The node checks to see whether the transaction changes data on itself or its descendants. If there is no change to the data, then the node skips the remaining steps and returns a read-only response (see "Read-Only Response" on page 32-12).

3. The node allocates the resources it needs to commit the transaction if data is changed.
4. The node saves redo records corresponding to changes made by the transaction to its redo log.
5. The node guarantees that locks held for the transaction are able to survive a failure.
6. The node responds to the initiating node with a prepared response (see "Prepared Response" on page 32-12) or, if its attempt or the attempt of one of its descendents to prepare was unsuccessful, with an abort response (see "Abort Response" on page 32-13).

These actions guarantee that the node can subsequently commit or roll back the transaction on the node. The prepared nodes then wait until a `COMMIT` or `ROLLBACK` request is received from the global coordinator.

After the nodes are prepared, the distributed transaction is said to be **in-doubt** (see "In-Doubt Transactions" on page 32-16).It retains in-doubt status until all changes are either committed or rolled back.

Commit Phase

The second phase in committing a distributed transaction is the commit phase. Before this phase occurs, *all* nodes other than the commit point site referenced in the distributed transaction have guaranteed that they are prepared, that is, they have the necessary resources to commit the transaction.

Steps in the Commit Phase

The commit phase consists of the following steps:

1. The global coordinator instructs the commit point site to commit.
2. The commit point site commits.
3. The commit point site informs the global coordinator that it has committed.
4. The global and local coordinators send a message to all nodes instructing them to commit the transaction.
5. At each node, the database commits the local portion of the distributed transaction and releases locks.
6. At each node, the database records an additional redo entry in the local redo log, indicating that the transaction has committed.

7. The participating nodes notify the global coordinator that they have committed.

When the commit phase is complete, the data on all nodes of the distributed system is consistent.

Guaranteeing Global Database Consistency

Each committed transaction has an associated system change number (SCN) to uniquely identify the changes made by the SQL statements within that transaction. The SCN functions as an internal timestamp that uniquely identifies a committed version of the database.

In a distributed system, the SCNs of communicating nodes are coordinated when all of the following actions occur:

- A connection occurs using the path described by one or more database links
- A distributed SQL statement executes
- A distributed transaction commits

Among other benefits, the coordination of SCNs among the nodes of a distributed system ensures global read-consistency at both the statement and transaction level. If necessary, global time-based recovery can also be completed.

During the prepare phase, the database determines the highest SCN at all nodes involved in the transaction. The transaction then commits with the high SCN at the commit point site. The commit SCN is then sent to all prepared nodes with the commit decision.

> **See Also:** "Managing Read Consistency" on page 33-25 for information about managing time lag issues in read consistency

Forget Phase

After the participating nodes notify the commit point site that they have committed, the commit point site can forget about the transaction. The following steps occur:

1. After receiving notice from the global coordinator that all nodes have committed, the commit point site erases status information about this transaction.
2. The commit point site informs the global coordinator that it has erased the status information.
3. The global coordinator erases its own information about the transaction.

In-Doubt Transactions

The two-phase commit mechanism ensures that all nodes either commit or perform a rollback together. What happens if any of the three phases fails because of a system or network error? The transaction becomes in-doubt.

Distributed transactions can become in-doubt in the following ways:

- A server machine running Oracle Database software crashes
- A network connection between two or more Oracle Databases involved in distributed processing is disconnected
- An unhandled software error occurs

The RECO process automatically resolves in-doubt transactions when the machine, network, or software problem is resolved. Until RECO can resolve the transaction, the data is locked for both reads and writes. The database blocks reads because it cannot determine which version of the data to display for a query.

This section contains the following topics:

- Automatic Resolution of In-Doubt Transactions
- Manual Resolution of In-Doubt Transactions
- Relevance of System Change Numbers for In-Doubt Transactions

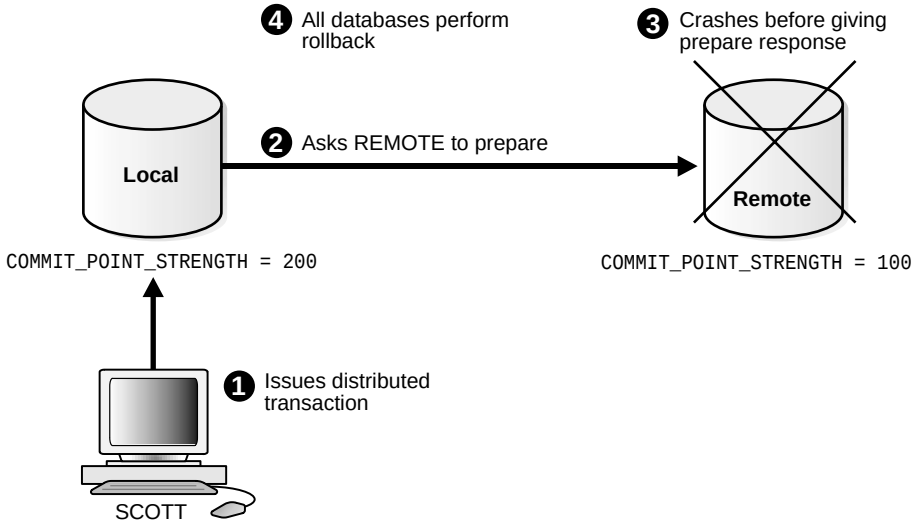
Automatic Resolution of In-Doubt Transactions

In the majority of cases, the database resolves the in-doubt transaction automatically. Assume that there are two nodes, `local` and `remote`, in the following scenarios. The local node is the commit point site. User `scott` connects to `local` and executes and commits a distributed transaction that updates `local` and `remote`.

Failure During the Prepare Phase

Figure 32–5 illustrates the sequence of events when there is a failure during the prepare phase of a distributed transaction:

Figure 32–5 Failure During Prepare Phase



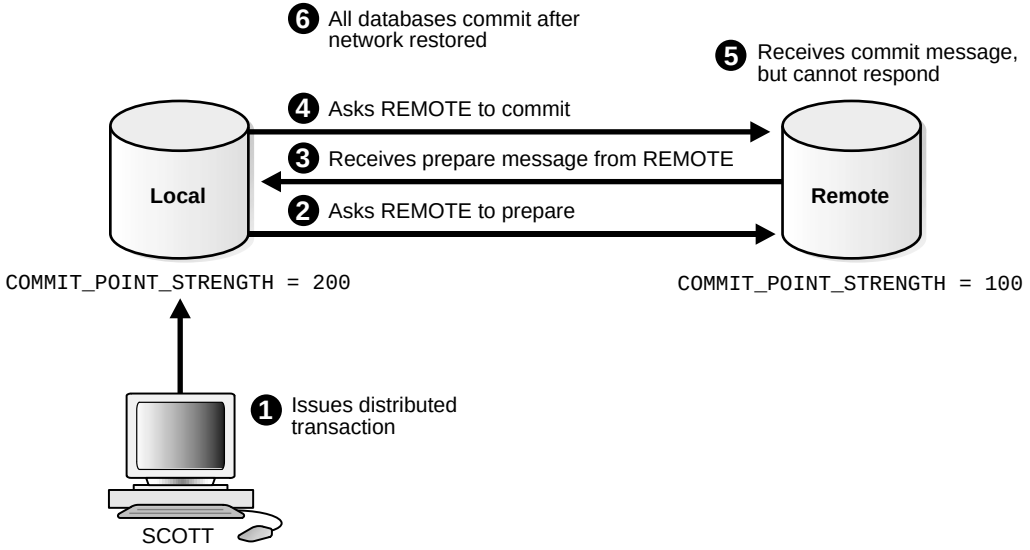
The following steps occur:

1. User `Scott` connects to `local` and executes a distributed transaction.
2. The global coordinator, which in this example is also the commit point site, requests all databases other than the commit point site to promise to commit or roll back when told to do so.
3. The `remote` database crashes before issuing the prepare response back to `local`.
4. The transaction is ultimately rolled back on each database by the RECO process when the remote site is restored.

Failure During the Commit Phase

Figure 32–6 illustrates the sequence of events when there is a failure during the commit phase of a distributed transaction:

Figure 32–6 Failure During Commit Phase



The following steps occur:

1. User `Scott` connects to `local` and executes a distributed transaction.
2. The global coordinator, which in this case is also the commit point site, requests all databases other than the commit point site to promise to commit or roll back when told to do so.
3. The commit point site receives a prepared message from `remote` saying that it will commit.
4. The commit point site commits the transaction locally, then sends a commit message to `remote` asking it to commit.
5. The `remote` database receives the commit message, but cannot respond because of a network failure.
6. The transaction is ultimately committed on the remote database by the RECO process after the network is restored.

> **See Also:** "Deciding How to Handle In-Doubt Transactions" on page 33-7 for a description of failure situations and how the database resolves intervening failures during two-phase commit

Manual Resolution of In-Doubt Transactions

You should only need to resolve an in-doubt transaction in the following cases:

- The in-doubt transaction has locks on critical data or undo segments.
- The cause of the machine, network, or software failure cannot be repaired quickly.

Resolution of in-doubt transactions can be complicated. The procedure requires that you do the following:

- Identify the transaction identification number for the in-doubt transaction.
- Query the `DBA_2PC_PENDING` and `DBA_2PC_NEIGHBORS` views to determine whether the databases involved in the transaction have committed.
- If necessary, force a commit using the `COMMIT FORCE` statement or a rollback using the `ROLLBACK FORCE` statement.

> **See Also:** The following sections explain how to resolve in-doubt transactions:
>
> - "Deciding How to Handle In-Doubt Transactions" on page 33-7
> - "Manually Overriding In-Doubt Transactions" on page 33-10

Relevance of System Change Numbers for In-Doubt Transactions

A **system change number** (SCN) is an internal timestamp for a committed version of the database. The Oracle Database server uses the SCN clock value to guarantee transaction consistency. For example, when a user commits a transaction, the database records an SCN for this commit in the redo log.

The database uses SCNs to coordinate distributed transactions among different databases. For example, the database uses SCNs in the following way:

1. An application establishes a connection using a database link.
2. The distributed transaction commits with the highest global SCN among all the databases involved.
3. The commit global SCN is sent to all databases involved in the transaction.

SCNs are important for distributed transactions because they function as a synchronized commit timestamp of a transaction, even if the transaction fails. If a transaction becomes in-doubt, an administrator can use this SCN to coordinate changes made to the global database. The global SCN for the transaction commit

can also be used to identify the transaction later, for example, in distributed recovery.

Distributed Transaction Processing: Case Study

In this scenario, a company has separate Oracle Database servers, `sales.acme.com` and `warehouse.acme.com`. As users insert sales records into the `sales` database, associated records are being updated at the `warehouse` database.

This case study of distributed processing illustrates:

- The definition of a session tree
- How a commit point site is determined
- When prepare messages are sent
- When a transaction actually commits
- What information is stored locally about the transaction

Stage 1: Client Application Issues DML Statements

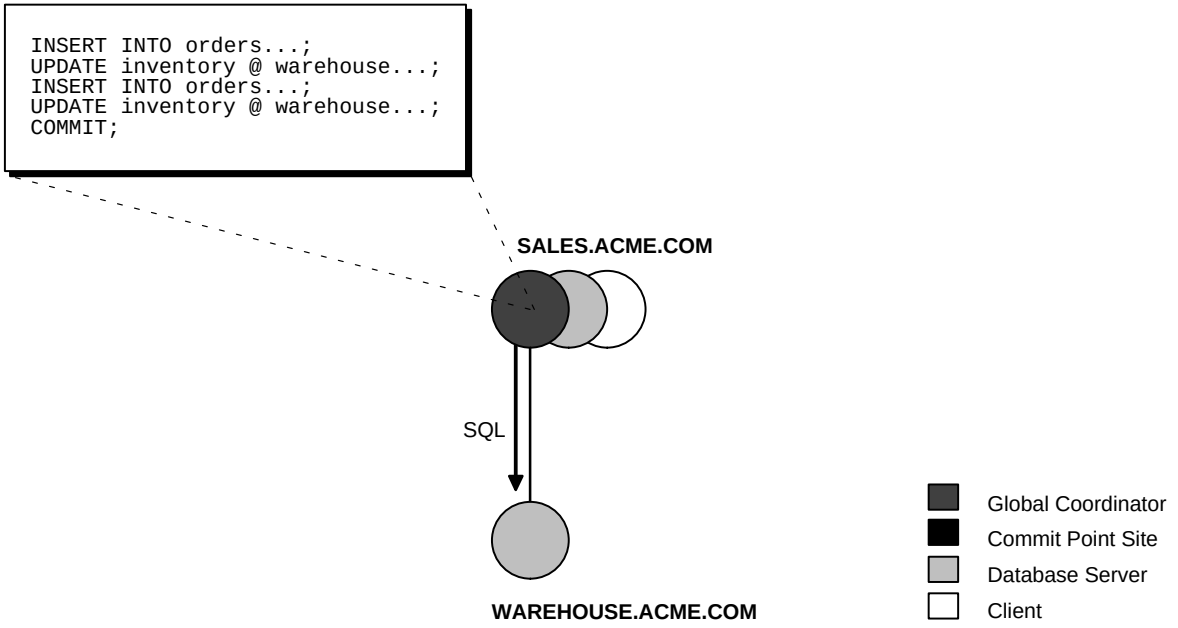
At the Sales department, a salesperson uses SQL*Plus to enter a sales order and then commit it. The application issues a number of SQL statements to enter the order into the `sales` database and update the inventory in the `warehouse` database:

```
CONNECT scott/tiger@sales.acme.com ...;
INSERT INTO orders ...;
UPDATE inventory@warehouse.acme.com ...;
INSERT INTO orders ...;
UPDATE inventory@warehouse.acme.com ...;
COMMIT;
```

These SQL statements are part of a single distributed transaction, guaranteeing that all issued SQL statements succeed or fail as a unit. Treating the statements as a unit prevents the possibility of an order being placed and then inventory not being updated to reflect the order. In effect, the transaction guarantees the consistency of data in the global database.

As each of the SQL statements in the transaction executes, the session tree is defined, as shown in Figure 32–7.

Figure 32–7 Defining the Session Tree



Note the following aspects of the transaction:

- An order entry application running on the `sales` database initiates the transaction. Therefore, `sales.acme.com` is the global coordinator for the distributed transaction.
- The order entry application inserts a new sales record into the `sales` database and updates the inventory at the warehouse. Therefore, the nodes `sales.acme.com` and `warehouse.acme.com` are both database servers.
- Because `sales.acme.com` updates the inventory, it is a client of `warehouse.acme.com`.

This stage completes the definition of the session tree for this distributed transaction. Each node in the tree has acquired the necessary data locks to execute the SQL statements that reference local data. These locks remain even after the SQL statements have been executed until the two-phase commit is completed.

Stage 2: Oracle Database Determines Commit Point Site

The database determines the commit point site immediately following the `COMMIT` statement. `sales.acme.com`, the global coordinator, is determined to be the commit point site, as shown in Figure 32–8.

> **See Also:** "Commit Point Strength" on page 32-8 for more information about how the commit point site is determined

Figure 32–8 Determining the Commit Point Site

Stage 3: Global Coordinator Sends Prepare Response

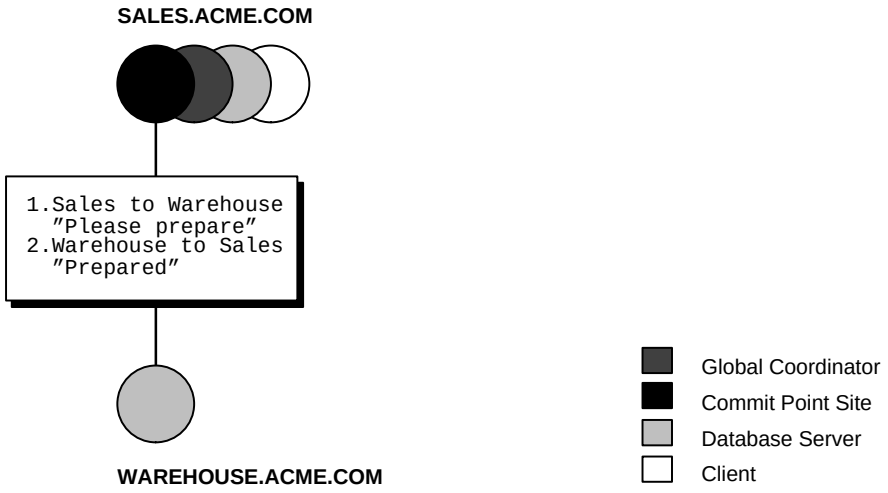
The prepare stage involves the following steps:

1. After the database determines the commit point site, the global coordinator sends the prepare message to all directly referenced nodes of the session tree, *excluding* the commit point site. In this example, `warehouse.acme.com` is the only node asked to prepare.
2. Node `warehouse.acme.com` tries to prepare. If a node can guarantee that it can commit the locally dependent part of the transaction and can record the commit information in its local redo log, then the node can successfully prepare. In this example, only `warehouse.acme.com` receives a prepare message because `sales.acme.com` is the commit point site.
3. Node `warehouse.acme.com` responds to `sales.acme.com` with a prepared message.

As each node prepares, it sends a message back to the node that asked it to prepare. Depending on the responses, one of the following can happen:

- If *any* of the nodes asked to prepare responds with an abort message to the global coordinator, then the global coordinator tells all nodes to roll back the transaction, and the operation is completed.
- If *all* nodes asked to prepare respond with a prepared or a read-only message to the global coordinator, that is, they have successfully prepared, then the global coordinator asks the commit point site to commit the transaction.

Figure 32–9 Sending and Acknowledging the Prepare Message



Stage 4: Commit Point Site Commits

The committing of the transaction by the commit point site involves the following steps:

1. Node `sales.acme.com`, receiving acknowledgment that `warehouse.acme.com` is prepared, instructs the commit point site to commit the transaction.
2. The commit point site now commits the transaction locally and records this fact in its local redo log.

Even if `warehouse.acme.com` has not yet committed, the outcome of this transaction is predetermined. In other words, the transaction *will* be committed at all nodes even if the ability of a given node to commit is delayed.

Stage 5: Commit Point Site Informs Global Coordinator of Commit

This stage involves the following steps:

1. The commit point site tells the global coordinator that the transaction has committed. Because the commit point site and global coordinator are the same node in this example, no operation is required. The commit point site knows that the transaction is committed because it recorded this fact in its online log.
2. The global coordinator confirms that the transaction has been committed on all other nodes involved in the distributed transaction.

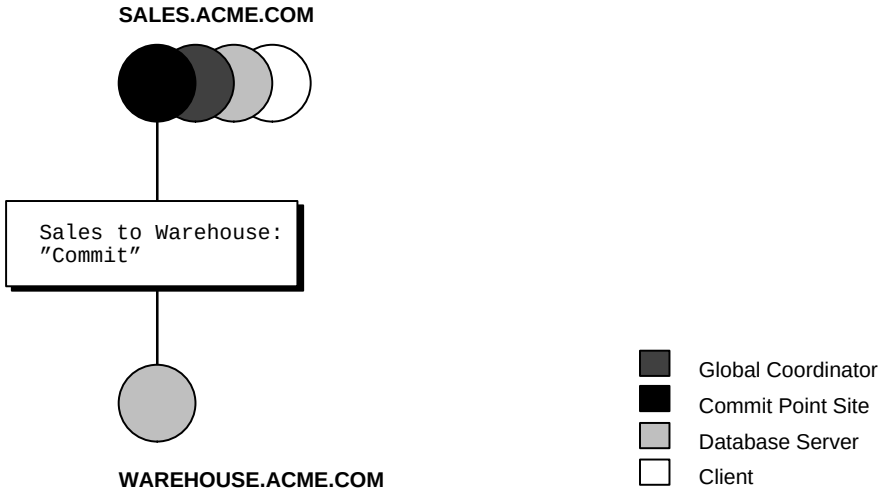
Stage 6: Global and Local Coordinators Tell All Nodes to Commit

The committing of the transaction by all the nodes in the transaction involves the following steps:

1. After the global coordinator has been informed of the commit at the commit point site, it tells all other directly referenced nodes to commit.
2. In turn, any local coordinators instruct their servers to commit, and so on.
3. Each node, including the global coordinator, commits the transaction and records appropriate redo log entries locally. As each node commits, the resource locks that were being held locally for that transaction are released.

In Figure 32–10, `sales.acme.com`, which is both the commit point site and the global coordinator, has already committed the transaction locally. `sales` now instructs `warehouse.acme.com` to commit the transaction.

Figure 32–10 Instructing Nodes to Commit



Stage 7: Global Coordinator and Commit Point Site Complete the Commit

The completion of the commit of the transaction occurs in the following steps:

1. After all referenced nodes and the global coordinator have committed the transaction, the global coordinator informs the commit point site of this fact.
2. The commit point site, which has been waiting for this message, erases the status information about this distributed transaction.
3. The commit point site informs the global coordinator that it is finished. In other words, the commit point site forgets about committing the distributed transaction. This action is permissible because all nodes involved in the two-phase commit have committed the transaction successfully, so they will never have to determine its status in the future.
4. The global coordinator finalizes the transaction by forgetting about the transaction itself.

After the completion of the `COMMIT` phase, the distributed transaction is itself complete. The steps described are accomplished automatically and in a fraction of a second.

33

Managing Distributed Transactions

This chapter describes how to manage and troubleshoot distributed transactions. The following topics are included in this chapter:

Specifying the Commit Point Strength of a Node

The database with the highest commit point strength determines which node commits first in a distributed transaction. When specifying a commit point strength for each node, ensure that the most critical server will be nonblocking if a failure occurs during a prepare or commit phase. The `COMMIT_POINT_STRENGTH` initialization parameter determines the commit point strength of a node.

The default value is operating system-dependent. The range of values is any integer from 0 to 255. For example, to set the commit point strength of a database to 200, include the following line in the database initialization parameter file:

```
COMMIT_POINT_STRENGTH = 200
```

The commit point strength is only used to determine the commit point site in a distributed transaction.

When setting the commit point strength for a database, note the following considerations:

- Because the commit point site stores information about the status of the transaction, the commit point site should not be a node that is frequently unreliable or unavailable in case other nodes need information about transaction status.
- Set the commit point strength for a database relative to the amount of critical shared data in the database. For example, a database on a mainframe computer usually shares more data among users than a database on a PC. Therefore, set the commit point strength of the mainframe to a higher value than the PC.

> **See Also:** "Commit Point Site" on page 32-6 for a conceptual overview of commit points

Naming Transactions

You can name a transaction. This is useful for identifying a specific distributed transaction and replaces the use of the `COMMIT COMMENT` statement for this purpose.

To name a transaction, use the `SET TRANSACTION ... NAME` statement. For example:

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
     NAME 'update inventory checkpoint 0';
```

This example shows that the user started a new transaction with isolation level equal to `SERIALIZABLE` and named it `'update inventory checkpoint 0'`.

For distributed transactions, the name is sent to participating sites when a transaction is committed. If a `COMMIT COMMENT` exists, it is ignored when a transaction name exists.

The transaction name is displayed in the `NAME` column of the `V$TRANSACTION` view, and in the `TRAN_COMMENT` field of the DBA_2PC_PENDING view when the transaction is committed.

Viewing Information About Distributed Transactions

The data dictionary of each database stores information about all open distributed transactions. You can use data dictionary tables and views to gain information about the transactions. This section contains the following topics:

- Determining the ID Number and Status of Prepared Transactions
- Tracing the Session Tree of In-Doubt Transactions

Determining the ID Number and Status of Prepared Transactions

The following view shows the database links that have been defined at the local database and stored in the data dictionary:

View	Purpose
DBA_2PC_PENDING	Lists all in-doubt distributed transactions. The view is empty until populated by an in-doubt transaction. After the transaction is resolved, the view is purged.

Use this view to determine the global commit number for a particular transaction ID. You can use this global commit number when manually resolving an in-doubt transaction.

The following table shows the most relevant columns (for a description of all the columns in the view, see *Oracle Database Reference*):

Table 33–1 DBA_2PC_PENDING

Column	Description
LOCAL_TRAN_ID	Local transaction identifier in the format *integer.integer.integer*. **Note:** When the LOCAL_TRAN_ID and the GLOBAL_TRAN_ID for a connection are the same, the node is the global coordinator of the transaction.
GLOBAL_TRAN_ID	Global database identifier in the format *global_db_name.db_hex_id.local_tran_id*, where *db_hex_id* is an eight-character hexadecimal value used to uniquely identify the database. This common transaction ID is the same on every node for a distributed transaction. **Note:** When the LOCAL_TRAN_ID and the GLOBAL_TRAN_ID for a connection are the same, the node is the global coordinator of the transaction.

Table 33–1 (Cont.) DBA_2PC_PENDING

Column	Description
STATE	STATE can have the following values: ■ Collecting This category normally applies only to the global coordinator or local coordinators. The node is currently collecting information from other database servers before it can decide whether it can prepare. ■ Prepared The node has prepared and may or may not have acknowledged this to its local coordinator with a prepared message. However, no commit request has been received. The node remains prepared, holding any local resource locks necessary for the transaction to commit. ■ Committed The node (any type) has committed the transaction, but other nodes involved in the transaction may not have done the same. That is, the transaction is still pending at one or more nodes. ■ Forced Commit A pending transaction can be forced to commit at the discretion of a database administrator. This entry occurs if a transaction is manually committed at a local node. ■ Forced termination (rollback) A pending transaction can be forced to roll back at the discretion of a database administrator. This entry occurs if this transaction is manually rolled back at a local node.
MIXED	YES means that part of the transaction was committed on one node and rolled back on another node.
TRAN_COMMENT	Transaction comment or, if using transaction naming, the transaction name is placed here when the transaction is committed.
HOST	Name of the host machine.
COMMIT#	Global commit number for committed transactions.

Execute the following script, named `pending_txn_script`, to query pertinent information in `DBA_2PC_PENDING` (sample output included):

```
COL LOCAL_TRAN_ID FORMAT A13
```

```
COL GLOBAL_TRAN_ID FORMAT A30
COL STATE FORMAT A8
COL MIXED FORMAT A3
COL HOST FORMAT A10
COL COMMIT# FORMAT A10

SELECT LOCAL_TRAN_ID, GLOBAL_TRAN_ID, STATE, MIXED, HOST, COMMIT#
   FROM DBA_2PC_PENDING
/

SQL> @pending_txn_script

LOCAL_TRAN_ID GLOBAL_TRAN_ID                  STATE    MIX HOST       COMMIT#
------------- ------------------------------ -------- --- ---------- ----------
1.15.870      HQ.ACME.COM.ef192da4.1.15.870  commit   no  dlsun183   115499
```

This output indicates that local transaction `1.15.870` has been committed on this node, but it may be pending on one or more other nodes. Because `LOCAL_TRAN_ID` and the local part of `GLOBAL_TRAN_ID` are the same, the node is the global coordinator of the transaction.

Tracing the Session Tree of In-Doubt Transactions

The following view shows which in-doubt transactions are incoming from a remote client and which are outgoing to a remote server:

View	Purpose
`DBA_2PC_NEIGHBORS`	Lists all incoming (from remote client) and outgoing (to remote server) in-doubt distributed transactions. It also indicates whether the local node is the commit point site in the transaction. The view is empty until populated by an in-doubt transaction. After the transaction is resolved, the view is purged.

When a transaction is in-doubt, you may need to determine which nodes performed which roles in the session tree. Use to this view to determine:

- All the incoming and outgoing connections for a given transaction
- Whether the node is the commit point site in a given transaction
- Whether the node is a global coordinator in a given transaction (because its local transaction ID and global transaction ID are the same)

The following table shows the most relevant columns (for an account of all the columns in the view, see *Oracle Database Reference*):

Table 33–2 DBA_2PC_NEIGHBORS

Column	Description
LOCAL_TRAN_ID	Local transaction identifier with the format *integer.integer.integer*. **Note:** When LOCAL_TRAN_ID and GLOBAL_TRAN_ID.DBA_2PC_PENDING for a connection are the same, the node is the global coordinator of the transaction.
IN_OUT	IN for incoming transactions; OUT for outgoing transactions.
DATABASE	For incoming transactions, the name of the client database that requested information from this local node; for outgoing transactions, the name of the database link used to access information on a remote server.
DBUSER_OWNER	For incoming transactions, the local account used to connect by the remote database link; for outgoing transactions, the owner of the database link.
INTERFACE	C is a commit message; N is either a message indicating a prepared state or a request for a read-only commit. When IN_OUT is OUT, C means that the child at the remote end of the connection is the commit point site and knows whether to commit or terminate. N means that the local node is informing the remote node that it is prepared. When IN_OUT is IN, C means that the local node or a database at the remote end of an outgoing connection is the commit point site. N means that the remote node is informing the local node that it is prepared.

Execute the following script, named neighbors_script, to query pertinent information in DBA_2PC_PENDING (sample output included):

```
COL LOCAL_TRAN_ID FORMAT A13
COL IN_OUT FORMAT A6
COL DATABASE FORMAT A25
COL DBUSER_OWNER FORMAT A15
COL INTERFACE FORMAT A3
SELECT LOCAL_TRAN_ID, IN_OUT, DATABASE, DBUSER_OWNER, INTERFACE
   FROM DBA_2PC_NEIGHBORS
/
```

```
SQL> CONNECT SYS/password@hq.acme.com
SQL> @neighbors_script

LOCAL_TRAN_ID IN_OUT DATABASE                  DBUSER_OWNER    INT
------------- ------ ------------------------- --------------- ---
1.15.870      out    SALES.ACME.COM            SYS             C
```

This output indicates that the local node sent an outgoing request to remote server `sales` to commit transaction `1.15.870`. If `sales` committed the transaction but no other node did, then you know that `sales` is the commit point site, because the commit point site always commits first.

Deciding How to Handle In-Doubt Transactions

A transaction is in-doubt when there is a failure during any aspect of the two-phase commit. Distributed transactions become in-doubt in the following ways:

- A server machine running Oracle Database software crashes
- A network connection between two or more Oracle Databases involved in distributed processing is disconnected
- An unhandled software error occurs

> **See Also:** "In-Doubt Transactions" on page 32-16 for a conceptual overview of in-doubt transactions

You can manually force the commit or rollback of a local, in-doubt distributed transaction. Because this operation can generate consistency problems, perform it only when specific conditions exist.

This section contains the following topics:

- Discovering Problems with a Two-Phase Commit
- Determining Whether to Perform a Manual Override
- Analyzing the Transaction Data

Discovering Problems with a Two-Phase Commit

The user application that commits a distributed transaction is informed of a problem by one of the following error messages:

```
ORA-02050: transaction ID rolled back,
           some remote dbs may be in-doubt
```

```
ORA-02053: transaction ID committed,
           some remote dbs may be in-doubt
ORA-02054: transaction ID in-doubt
```

A robust application should save information about a transaction if it receives any of the preceding errors. This information can be used later if manual distributed transaction recovery is desired.

No action is required by the administrator of any node that has one or more in-doubt distributed transactions due to a network or system failure. The automatic recovery features of the database transparently complete any in-doubt transaction so that the same outcome occurs on all nodes of a session tree (that is, all commit or all roll back) after the network or system failure is resolved.

In extended outages, however, you can force the commit or rollback of a transaction to release any locked data. Applications must account for such possibilities.

Determining Whether to Perform a Manual Override

Override a specific in-doubt transaction manually *only* when one of the following conditions exists:

- The in-doubt transaction locks data that is required by other transactions. This situation occurs when the `ORA-01591` error message interferes with user transactions.
- An in-doubt transaction prevents the extents of a undo segment from being used by other transactions. The first portion of the local transaction ID of an in-doubt distributed transaction corresponds to the ID of the undo segment, as listed by the data dictionary view `DBA_2PC_PENDING`.
- The failure preventing the two-phase commit phases to complete cannot be corrected in an acceptable time period. Examples of such cases include a telecommunication network that has been damaged or a damaged database that requires a long recovery time.

Normally, you should make a decision to locally force an in-doubt distributed transaction in consultation with administrators at other locations. A wrong decision can lead to database inconsistencies that can be difficult to trace and that you must manually correct.

If none of these conditions apply, *always* allow the automatic recovery features of the database to complete the transaction. If any of these conditions are met, however, consider a local override of the in-doubt transaction.

Analyzing the Transaction Data

If you decide to force the transaction to complete, analyze available information with the following goals in mind.

Find a Node that Committed or Rolled Back

Use the DBA_2PC_PENDING view to find a node that has either committed or rolled back the transaction. If you can find a node that has already resolved the transaction, then you can follow the action taken at that node.

Look for Transaction Comments

See if any information is given in the TRAN_COMMENT column of DBA_2PC_PENDING for the distributed transaction. Comments are included in the COMMENT clause of the COMMIT statement, or if transaction naming is used, the transaction name is placed in the TRAN_COMMENT field when the transaction is committed.

For example, the comment of an in-doubt distributed transaction can indicate the origin of the transaction and what type of transaction it is:

```
COMMIT COMMENT 'Finance/Accts_pay/Trans_type 10B';
```

The SET TRANSACTION ... NAME statement could also have been used (and is preferable) to provide this information in a transaction name.

See Also: "Naming Transactions" on page 33-2

Look for Transaction Advice

See if any information is given in the ADVICE column of DBA_2PC_PENDING for the distributed transaction. An application can prescribe advice about whether to force the commit or force the rollback of separate parts of a distributed transaction with the ADVISE clause of the ALTER SESSION statement.

The advice sent during the prepare phase to each node is the advice in effect at the time the most recent DML statement executed at that database in the current transaction.

For example, consider a distributed transaction that moves an employee record from the emp table at one node to the emp table at another node. The transaction can protect the record--even when administrators independently force the in-doubt transaction at each node--by including the following sequence of SQL statements:

```
ALTER SESSION ADVISE COMMIT;
INSERT INTO emp@hq ... ;    /*advice to commit at HQ */
```

```
ALTER SESSION ADVISE ROLLBACK;
DELETE FROM emp@sales ... ; /*advice to roll back at SALES*/

ALTER SESSION ADVISE NOTHING;
```

If you manually force the in-doubt transaction following the given advice, the worst that can happen is that each node has a copy of the employee record; the record cannot disappear.

Manually Overriding In-Doubt Transactions

Use the COMMIT or ROLLBACK statement with the FORCE option and a text string that indicates either the local or global transaction ID of the in-doubt transaction to commit.

> **Note:** In all examples, the transaction is committed or rolled back on the local node, and the local pending transaction table records a value of forced commit or forced termination for the STATE column the row for this transaction.

This section contains the following topics:

- Manually Committing an In-Doubt Transaction
- Manually Rolling Back an In-Doubt Transaction

Manually Committing an In-Doubt Transaction

Before attempting to commit the transaction, ensure that you have the proper privileges. Note the following requirements:

User Committing the Transaction	Privilege Required
You	FORCE TRANSACTION
Another user	FORCE ANY TRANSACTION

Committing Using Only the Transaction ID

The following SQL statement commits an in-doubt transaction:

```
COMMIT FORCE 'transaction_id';
```

The variable *transaction_id* is the identifier of the transaction as specified in either the LOCAL_TRAN_ID or GLOBAL_TRAN_ID columns of the DBA_2PC_PENDING data dictionary view.

For example, assume that you query DBA_2PC_PENDING and determine that LOCAL_TRAN_ID for a distributed transaction is 1:45.13.

You then issue the following SQL statement to force the commit of this in-doubt transaction:

```
COMMIT FORCE '1.45.13';
```

Committing Using an SCN

Optionally, you can specify the SCN for the transaction when forcing a transaction to commit. This feature lets you commit an in-doubt transaction with the SCN assigned when it was committed at other nodes.

Consequently, you maintain the synchronized commit time of the distributed transaction even if there is a failure. Specify an SCN only when you can determine the SCN of the same transaction already committed at another node.

For example, assume you want to manually commit a transaction with the following global transaction ID:

```
SALES.ACME.COM.55d1c563.1.93.29
```

First, query the DBA_2PC_PENDING view of a remote database also involved with the transaction in question. Note the SCN used for the commit of the transaction at that node. Specify the SCN when committing the transaction at the local node. For example, if the SCN is 829381993, issue:

```
COMMIT FORCE 'SALES.ACME.COM.55d1c563.1.93.29', 829381993;
```

> **See Also:** *Oracle Database SQL Reference* for more information about using the COMMIT statement

Manually Rolling Back an In-Doubt Transaction

Before attempting to roll back the in-doubt distributed transaction, ensure that you have the proper privileges. Note the following requirements:

User Committing the Transaction	Privilege Required
You	FORCE TRANSACTION

User Committing the Transaction	Privilege Required
Another user	FORCE ANY TRANSACTION

The following SQL statement rolls back an in-doubt transaction:

```
ROLLBACK FORCE 'transaction_id';
```

The variable *transaction_id* is the identifier of the transaction as specified in either the LOCAL_TRAN_ID or GLOBAL_TRAN_ID columns of the DBA_2PC_PENDING data dictionary view.

For example, to roll back the in-doubt transaction with the local transaction ID of 2.9.4, use the following statement:

```
ROLLBACK FORCE '2.9.4';
```

Note: You cannot roll back an in-doubt transaction to a savepoint.

See Also: *Oracle Database SQL Reference* for more information about using the ROLLBACK statement

Purging Pending Rows from the Data Dictionary

Before RECO recovers an in-doubt transaction, the transaction appears in DBA_2PC_PENDING.STATE as COLLECTING, COMMITTED, or PREPARED. If you force an in-doubt transaction using COMMIT FORCE or ROLLBACK FORCE, then the states FORCED COMMIT or FORCED ROLLBACK may appear.

Automatic recovery normally deletes entries in these states. The only exception is when recovery discovers a forced transaction that is in a state inconsistent with other sites in the transaction. In this case, the entry can be left in the table and the MIXED column in DBA_2PC_PENDING has a value of YES. These entries can be cleaned up with the DBMS_TRANSACTION.PURGE_MIXED procedure.

If automatic recovery is not possible because a remote database has been permanently lost, then recovery cannot identify the re-created database because it receives a new database ID when it is re-created. In this case, you must use the PURGE_LOST_DB_ENTRY procedure in the DBMS_TRANSACTION package to clean up the entries. The entries do not hold up database resources, so there is no urgency in cleaning them up.

See Also: *PL/SQL Packages and Types Reference* for more information about the DBMS_TRANSACTION package

Executing the PURGE_LOST_DB_ENTRY Procedure

To manually remove an entry from the data dictionary, use the following syntax (where *trans_id* is the identifier for the transaction):

```
DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY('trans_id');
```

For example, to purge pending distributed transaction `1.44.99`, enter the following statement in SQL*Plus:

```
EXECUTE DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY('1.44.99');
```

Execute this procedure only if significant reconfiguration has occurred so that automatic recovery cannot resolve the transaction. Examples include:

- Total loss of the remote database
- Reconfiguration in software resulting in loss of two-phase commit capability
- Loss of information from an external transaction coordinator such as a TPMonitor

Determining When to Use DBMS_TRANSACTION

The following tables indicates what the various states indicate about the distributed transaction what the administrator's action should be:

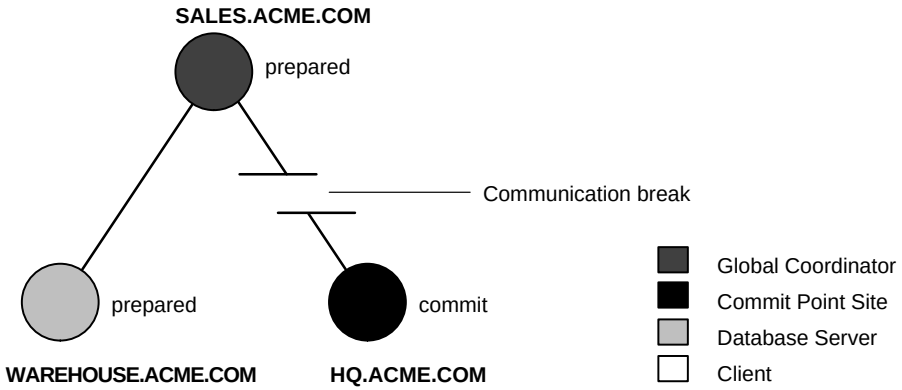
STATE Column	State of Global Transaction	State of Local Transaction	Normal Action	Alternative Action
Collecting	Rolled back	Rolled back	None	PURGE_LOST_DB_ ENTRY (only if autorecovery cannot resolve transaction)
Committed	Committed	Committed	None	PURGE_LOST_DB_ ENTRY (only if autorecovery cannot resolve transaction)
Prepared	Unknown	Prepared	None	Force commit or rollback

STATE Column	State of Global Transaction	State of Local Transaction	Normal Action	Alternative Action
Forced commit	Unknown	Committed	None	PURGE_LOST_DB_ENTRY (only if autorecovery cannot resolve transaction)
Forced rollback	Unknown	Rolled back	None	PURGE_LOST_DB_ENTRY (only if autorecovery cannot resolve transaction)
Forced commit	Mixed	Committed	Manually remove inconsistencies then use PURGE_MIXED	-
Forced rollback	Mixed	Rolled back	Manually remove inconsistencies then use PURGE_MIXED	-

Manually Committing an In-Doubt Transaction: Example

Figure 33–1, illustrates a failure during the commit of a distributed transaction. In this failure case, the prepare phase completes. During the commit phase, however, the commit confirmation of the commit point site never reaches the global coordinator, even though the commit point site committed the transaction. Inventory data is locked and cannot be accessed because the in-doubt transaction is critical to other transactions. Further, the locks must be held until the in-doubt transaction either commits or rolls back.

Figure 33–1 Example of an In-Doubt Distributed Transaction



You can manually force the local portion of the in-doubt transaction by following the steps detailed in the following sections:

Step 1: Record User Feedback

Step 2: Query DBA_2PC_PENDING

Step 3: Query DBA_2PC_NEIGHBORS on Local Node

Step 4: Querying Data Dictionary Views on All Nodes

Step 5: Commit the In-Doubt Transaction

Step 6: Check for Mixed Outcome Using DBA_2PC_PENDING

Step 1: Record User Feedback

The users of the local database system that conflict with the locks of the in-doubt transaction receive the following error message:

```
ORA-01591: lock held by in-doubt distributed transaction 1.21.17
```

In this case, `1.21.17` is the local transaction ID of the in-doubt distributed transaction. You should request and record this ID number from users that report problems to identify which in-doubt transactions should be forced.

Step 2: Query DBA_2PC_PENDING

After connecting with SQL*Plus to `warehouse`, query the local `DBA_2PC_PENDING` data dictionary view to gain information about the in-doubt transaction:

```
CONNECT SYS/password@warehouse.acme.com
SELECT * FROM DBA_2PC_PENDING WHERE LOCAL_TRAN_ID = '1.21.17';
```

The database returns the following information:

```
Column Name            Value
---------------------- -------------------------------------
LOCAL_TRAN_ID          1.21.17
GLOBAL_TRAN_ID         SALES.ACME.COM.55d1c563.1.93.29
STATE                  prepared
MIXED                  no
ADVICE
TRAN_COMMENT           Sales/New Order/Trans_type 10B
FAIL_TIME              31-MAY-91
FORCE_TIME
RETRY_TIME             31-MAY-91
OS_USER                SWILLIAMS
OS_TERMINAL            TWA139:
HOST                   system1
DB_USER                SWILLIAMS
COMMIT#
```

Determining the Global Transaction ID

The global transaction ID is the common transaction ID that is the same on every node for a distributed transaction. It is of the form:

global_database_name.hhhhhhhh.local_transaction_id

where:

- *global_database_name* is the database name of the global coordinator.
- *hhhhhhhh* is the internal database identifier of the global coordinator (in hexadecimal).
- *local_transaction_id* is the corresponding local transaction ID assigned on the global coordinator.

Note that the last portion of the global transaction ID and the local transaction ID match at the global coordinator. In the example, you can tell that `warehouse` is *not* the global coordinator because these numbers do not match:

```
LOCAL_TRAN_ID          1.21.17
GLOBAL_TRAN_ID         ... 1.93.29
```

Determining the State of the Transaction

The transaction on this node is in a prepared state:

```
STATE          prepared
```

Therefore, warehouse waits for its coordinator to send either a commit or a rollback request.

Looking for Comments or Advice

The transaction comment or advice can include information about this transaction. If so, use this comment to your advantage. In this example, the origin and transaction type is in the transaction comment:

```
TRAN_COMMENT           Sales/New Order/Trans_type 10B
```

It could also be provided as a transaction name with a SET TRANSACTION ... NAME statement.

This information can reveal something that helps you decide whether to commit or rollback the local portion of the transaction. If useful comments do not accompany an in-doubt transaction, you must complete some extra administrative work to trace the session tree and find a node that has resolved the transaction.

Step 3: Query DBA_2PC_NEIGHBORS on Local Node

The purpose of this step is to climb the session tree so that you find coordinators, eventually reaching the global coordinator. Along the way, you may find a coordinator that has resolved the transaction. If not, you can eventually work your way to the commit point site, which will always have resolved the in-doubt transaction. To trace the session tree, query the DBA_2PC_NEIGHBORS view on each node.

In this case, you query this view on the warehouse database:

```
CONNECT SYS/password@warehouse.acme.com
SELECT * FROM DBA_2PC_NEIGHBORS
  WHERE LOCAL_TRAN_ID = '1.21.17'
  ORDER BY SESS#, IN_OUT;

Column Name            Value
---------------------- -------------------------------------
LOCAL_TRAN_ID          1.21.17
IN_OUT                 in
DATABASE               SALES.ACME.COM
```

```
DBUSER_OWNER            SWILLIAMS
INTERFACE               N
DBID                    000003F4
SESS#                   1
BRANCH                  0100
```

Obtaining Database Role and Database Link Information

The DBA_2PC_NEIGHBORS view provides information about connections associated with an in-doubt transaction. Information for each connection is different, based on whether the connection is **inbound** (IN_OUT = in) or **outbound** (IN_OUT = out):

IN_OUT	Meaning	DATABASE	DBUSER_OWNER
in	Your node is a server of another node.	Lists the name of the client database that connected to your node.	Lists the local account for the database link connection that corresponds to the in-doubt transaction.
out	Your node is a client of other servers.	Lists the name of the database link that connects to the remote node.	Lists the owner of the database link for the in-doubt transaction.

In this example, the IN_OUT column reveals that the warehouse database is a server for the sales client, as specified in the DATABASE column:

```
IN_OUT                  in
DATABASE                SALES.ACME.COM
```

The connection to warehouse was established through a database link from the swilliams account, as shown by the DBUSER_OWNER column:

```
DBUSER_OWNER            SWILLIAMS
```

Determining the Commit Point Site

Additionally, the INTERFACE column tells whether the local node or a subordinate node is the commit point site:

```
INTERFACE               N
```

Neither warehouse nor any of its descendants is the commit point site, as shown by the INTERFACE column.

Step 4: Querying Data Dictionary Views on All Nodes

At this point, you can contact the administrator at the located nodes and ask each person to repeat Steps 2 and 3 using the global transaction ID.

> **Note:** If you can directly connect to these nodes with another network, you can repeat Steps 2 and 3 yourself.

For example, the following results are returned when Steps 2 and 3 are performed at `sales` and `hq`.

Checking the Status of Pending Transactions at sales

At this stage, the `sales` administrator queries the `DBA_2PC_PENDING` data dictionary view:

```
SQL> CONNECT SYS/password@sales.acme.com
SQL> SELECT * FROM DBA_2PC_PENDING
   > WHERE GLOBAL_TRAN_ID = 'SALES.ACME.COM.55d1c563.1.93.29';

Column Name            Value
---------------------- -------------------------------------
LOCAL_TRAN_ID          1.93.29
GLOBAL_TRAN_ID         SALES.ACME.COM.55d1c563.1.93.29
STATE                  prepared
MIXED                  no
ADVICE
TRAN_COMMENT           Sales/New Order/Trans_type 10B
FAIL_TIME              31-MAY-91
FORCE_TIME
RETRY_TIME             31-MAY-91
OS_USER                SWILLIAMS
OS_TERMINAL            TWA139:
HOST                   system1
DB_USER                SWILLIAMS
COMMIT#
```

Determining the Coordinators and Commit Point Site at sales

Next, the `sales` administrator queries `DBA_2PC_NEIGHBORS` to determine the global and local coordinators as well as the commit point site:

```
SELECT * FROM DBA_2PC_NEIGHBORS
   WHERE GLOBAL_TRAN_ID = 'SALES.ACME.COM.55d1c563.1.93.29'
```

```
   ORDER BY SESS#, IN_OUT;
```

This query returns three rows:

- The connection to `warehouse`
- The connection to `hq`
- The connection established by the user

Reformatted information corresponding to the rows for the `warehouse` connection appears below:

```
Column Name          Value
-------------------- -----------------------------------
LOCAL_TRAN_ID        1.93.29
IN_OUT               OUT
DATABASE             WAREHOUSE.ACME.COM
DBUSER_OWNER         SWILLIAMS
INTERFACE            N
DBID                 55d1c563
SESS#                1
BRANCH               1
```

Reformatted information corresponding to the rows for the `hq` connection appears below:

```
Column Name          Value
-------------------- -----------------------------------
LOCAL_TRAN_ID        1.93.29
IN_OUT               OUT
DATABASE             HQ.ACME.COM
DBUSER_OWNER         ALLEN
INTERFACE            C
DBID                 00000390
SESS#                1
BRANCH               1
```

The information from the previous queries reveal the following:

- `sales` is the global coordinator because the local transaction ID and global transaction ID match.
- Two outbound connections are established from this node, but no inbound connections. `sales` is not the server of another node.
- `hq` or one of its servers is the commit point site.

Checking the Status of Pending Transactions at HQ

At this stage, the hq administrator queries the DBA_2PC_PENDING data dictionary view:

```
SELECT * FROM DBA_2PC_PENDING@hq.acme.com
   WHERE GLOBAL_TRAN_ID = 'SALES.ACME.COM.55d1c563.1.93.29';

Column Name           Value
--------------------- ------------------------------------
LOCAL_TRAN_ID         1.45.13
GLOBAL_TRAN_ID        SALES.ACME.COM.55d1c563.1.93.29
STATE                 COMMIT
MIXED                 NO
ACTION
TRAN_COMMENT          Sales/New Order/Trans_type 10B
FAIL_TIME             31-MAY-91
FORCE_TIME
RETRY_TIME            31-MAY-91
OS_USER               SWILLIAMS
OS_TERMINAL           TWA139:
HOST                  SYSTEM1
DB_USER               SWILLIAMS
COMMIT#               129314
```

At this point, you have found a node that resolved the transaction. As the view reveals, it has been committed and assigned a commit ID number:

```
STATE                 COMMIT
COMMIT#               129314
```

Therefore, you can force the in-doubt transaction to commit at your local database. It is a good idea to contact any other administrators you know that could also benefit from your investigation.

Step 5: Commit the In-Doubt Transaction

You contact the administrator of the sales database, who manually commits the in-doubt transaction using the global ID:

```
SQL> CONNECT SYS/password@sales.acme.com
SQL> COMMIT FORCE 'SALES.ACME.COM.55d1c563.1.93.29';
```

As administrator of the warehouse database, you manually commit the in-doubt transaction using the global ID:

```
SQL> CONNECT SYS/password@warehouse.acme.com
SQL> COMMIT FORCE 'SALES.ACME.COM.55d1c563.1.93.29';
```

Step 6: Check for Mixed Outcome Using DBA_2PC_PENDING

After you manually force a transaction to commit or roll back, the corresponding row in the pending transaction table remains. The state of the transaction is changed depending on how you forced the transaction.

Every Oracle Database has a **pending transaction table**. This is a special table that stores information about distributed transactions as they proceed through the two-phase commit phases. You can query the pending transaction table of a database through the `DBA_2PC_PENDING` data dictionary view (see Table 33–1).

Also of particular interest in the pending transaction table is the mixed outcome flag as indicated in `DBA_2PC_PENDING.MIXED`. You can make the wrong choice if a pending transaction is forced to commit or roll back. For example, the local administrator rolls back the transaction, but the other nodes commit it. Incorrect decisions are detected automatically, and the damage flag for the corresponding pending transaction record is set (`MIXED=yes`).

The RECO (Recoverer) background process uses the information in the pending transaction table to finalize the status of in-doubt transactions. You can also use the information in the pending transaction table to manually override the automatic recovery procedures for pending distributed transactions.

All transactions automatically resolved by RECO are removed from the pending transaction table. Additionally, all information about in-doubt transactions correctly resolved by an administrator (as checked when RECO reestablishes communication) are automatically removed from the pending transaction table. However, all rows resolved by an administrator that result in a mixed outcome across nodes remain in the pending transaction table of all involved nodes until they are manually deleted using `DBMS_TRANSACTIONS.PURGE_MIXED`.

Data Access Failures Due to Locks

When you issue a SQL statement, the database attempts to lock the resources needed to successfully execute the statement. If the requested data is currently held by statements of other uncommitted transactions, however, and remains locked for a long time, a timeout occurs.

Consider the following scenarios involving data access failure:

- Transaction Timeouts

Transaction Timeouts

A DML statement that requires locks on a remote database can be blocked if another transaction own locks on the requested data. If these locks continue to block the requesting SQL statement, then the following sequence of events occurs:

1. A timeout occurs.
2. The database rolls back the statement.
3. The database returns this error message to the user:

```
ORA-02049: time-out: distributed transaction waiting for lock
```

Because the transaction did not modify data, no actions are necessary as a result of the timeout. Applications should proceed as if a deadlock has been encountered. The user who executed the statement can try to reexecute the statement later. If the lock persists, then the user should contact an administrator to report the problem.

Locks from In-Doubt Transactions

A query or DML statement that requires locks on a local database can be blocked indefinitely due to the locked resources of an in-doubt distributed transaction. In this case, the database issues the following error message:

```
ORA-01591: lock held by in-doubt distributed transaction identifier
```

In this case, the database rolls back the SQL statement immediately. The user who executed the statement can try to reexecute the statement later. If the lock persists, the user should contact an administrator to report the problem, *including* the ID of the in-doubt distributed transaction.

The chances of these situations occurring are rare considering the low probability of failures during the critical portions of the two-phase commit. Even if such a failure occurs, and assuming quick recovery from a network or system failure, problems are automatically resolved without manual intervention. Thus, problems usually resolve before they can be detected by users or database administrators.

Simulating Distributed Transaction Failure

You can force the failure of a distributed transaction for the following reasons:

- To observe RECO automatically resolving the local portion of the transaction

- To practice manually resolving in-doubt distributed transactions and observing the results

This section describes the features available and the steps necessary to perform such operations.

Forcing a Distributed Transaction to Fail

You can include comments in the COMMENT parameter of the COMMIT statement. To intentionally induce a failure during the two-phase commit phases of a distributed transaction, include the following comment in the COMMENT parameter:

```
COMMIT COMMENT 'ORA-2PC-CRASH-TEST-n';
```

where *n* is one of the following integers:

n	Effect
1	Crash commit point after collect
2	Crash non-commit-point site after collect
3	Crash before prepare (non-commit-point site)
4	Crash after prepare (non-commit-point site)
5	Crash commit point site before commit
6	Crash commit point site after commit
7	Crash non-commit-point site before commit
8	Crash non-commit-point site after commit
9	Crash commit point site before forget
10	Crash non-commit-point site before forget

For example, the following statement returns the following messages if the local commit point strength is greater than the remote commit point strength and both nodes are updated:

```
COMMIT COMMENT 'ORA-2PC-CRASH-TEST-7';

ORA-02054: transaction 1.93.29 in-doubt
ORA-02059: ORA_CRASH_TERST_7 in commit comment
```

At this point, the in-doubt distributed transaction appears in the DBA_2PC_PENDING view. If enabled, RECO automatically resolves the transaction.

Disabling and Enabling RECO

The RECO background process of an Oracle Database instance automatically resolves failures involving distributed transactions. At exponentially growing time intervals, the RECO background process of a node attempts to recover the local portion of an in-doubt distributed transaction.

RECO can use an existing connection or establish a new connection to other nodes involved in the failed transaction. When a connection is established, RECO automatically resolves all in-doubt transactions. Rows corresponding to any resolved in-doubt transactions are automatically removed from the pending transaction table of each database.

You can enable and disable RECO using the ALTER SYSTEM statement with the ENABLE/DISABLE DISTRIBUTED RECOVERY options. For example, you can temporarily disable RECO to force the failure of a two-phase commit and manually resolve the in-doubt transaction.

The following statement disables RECO:

```
ALTER SYSTEM DISABLE DISTRIBUTED RECOVERY;
```

Alternatively, the following statement enables RECO so that in-doubt transactions are automatically resolved:

```
ALTER SYSTEM ENABLE DISTRIBUTED RECOVERY;
```

Note: Single-process instances (for example, a PC running MS-DOS) have no separate background processes, and therefore no RECO process. Therefore, when a single-process instance that participates in a distributed system is started, you must manually enable distributed recovery using the preceding statement.

Managing Read Consistency

An important restriction exists in the Oracle Database implementation of distributed read consistency. The problem arises because each system has its own SCN, which you can view as the database internal timestamp. The Oracle Database server uses the SCN to decide which version of data is returned from a query.

The SCNs in a distributed transaction are synchronized at the end of each remote SQL statement and at the start and end of each transaction. Between two nodes that have heavy traffic and especially distributed updates, the synchronization is frequent. Nevertheless, no practical way exists to keep SCNs in a distributed system absolutely synchronized: a window always exists in which one node may have an SCN that is somewhat in the past with respect to the SCN of another node.

Because of the SCN gap, you can execute a query that uses a slightly old snapshot, so that the most recent changes to the remote database are not seen. In accordance with read consistency, a query can therefore retrieve consistent, but out-of-date data. Note that all data retrieved by the query will be from the old SCN, so that if a locally executed update transaction updates two tables at a remote node, then data selected from both tables in the next remote access contain data prior to the update.

One consequence of the SCN gap is that two consecutive `SELECT` statements can retrieve different data even though no DML has been executed between the two statements. For example, you can issue an update statement and then commit the update on the remote database. When you issue a `SELECT` statement on a view based on this remote table, the view does not show the update to the row. The next time that you issue the `SELECT` statement, the update is present.

You can use the following techniques to ensure that the SCNs of the two machines are synchronized just before a query:

- Because SCNs are synchronized at the end of a remote query, precede each remote query with a dummy remote query to the same site, for example, `SELECT * FROM DUAL@REMOTE`.
- Because SCNs are synchronized at the start of every remote transaction, commit or roll back the current transaction before issuing the remote query.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W