Oracle8*i*

Administrator's Guide

Release 8.1.5

February 1999

Part No. A67772-01

ORACLE

Administrator's Guide, Release 8.1.5

Part No. A67772-01

Primary Author: Joyce Fee

Contributing Authors: Alex Tsukerman, Andre Kruglikov, Ann Rhee, Ashwini Surpur, Bhaskar Himatsingka, Harvey Eneman, Jags Srinivasan, Lois Price, Robert Jenkins, Sophia Yeung, Vinay Srihari, Wei Huang, Jonathan Klein, Mike Hartstein, Bill Lee, Diana Lorentz, Lance Ashdown, Phil Locke, Ekrem Soylemez, Connie Dialaris, Steven Wertheimer, Val Kane, Mary Rhodes, Archna Kalra, Nina Lewis

Graphic Designer: Valarie Moore

Contents

2 Creating an Oracle Database

4 Managing Oracle Processes

5 Managing Control Files

6 Managing the Online Redo Log

7 Managing Archived Redo Logs

8 Managing Job Queues

10 Managing Datafiles

11 Using the Database Resource Manager

12 Guidelines for Managing Schema Objects

13 Managing Partitioned Tables and Indexes

14 Managing Tables

15 Managing Views, Sequences and Synonyms

16 Managing Indexes

17 Managing Clusters

20 General Management of Schema Objects

21 Managing Rollback Segments

24 Managing User Privileges and Roles

25 Auditing Database Use

Index

Send Us Your Comments

Administrator's Guide, Release 8.1.5

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 chapter, section, and page number (if available). Please send your comments to:

Server Technologies Documentation Manager
Oracle Corporation
500 Oracle Parkway
Redwood Shores, CA 94065
Fax: (650) 506-7228

or e-mail comments to the Information Development department at the following e-mail address:

infodev@us.oracle.com

Preface

This guide is for people who administer the operation of an Oracle database system. These people, referred to as "database administrators" (DBAs), are assumed to be responsible for ensuring the smooth operation of an Oracle database system and for monitoring its use. The responsibilities of database administrators are described in Chapter 1.

Attention: The *Oracle8i Administrator's Guide* contains information that describes the features and functionality of the Oracle8 and the Oracle8 Enterprise Edition products. Oracle8 and Oracle8 Enterprise Edition have the same basic features. However, several advanced features are available only with the Enterprise Edition, and some of these are optional. For example, to perform automated tablespace point-in-time recovery (using Recovery Manager), you must have the Enterprise Edition.

For information about the differences between Oracle8 and the Oracle8 Enterprise Edition and the features and options that are available to you, please refer to *Getting to Know Oracle8i and the Oracle8i Enterprise Edition.*

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.

Readers Interested in Installation and Migration Information

Administrators frequently participate in installing the Oracle Server software and migrating existing Oracle databases to newer formats (for example, Version 7 databases to Oracle8 format). This guide is not an installation or migration manual.

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

If your primary interest is database or application migration, see the *Oracle8i Migration* manual.

Readers Interested in Application Design Information

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

However, database application developers should also see the *Oracle8i Application Developer's Guide - Fundamentals* and the documentation for the tool or language product they are using to develop Oracle database applications.

How to Use This Guide

Every reader of this guide should read Chapter 1 of *Oracle8i Concepts*. This overview of the concepts and terminology related to Oracle provides a foundation for the more detailed information in this guide. The rest of *Oracle8i Concepts* explains the Oracle architecture and features, and how they operate in more detail.

Structure

This guide contains the following parts and chapters.

Part I: Basic Database Administration

Chapter 1, "The Oracle Database Administrator"	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 describes the most important considerations when creating a database. Consult this chapter when in the database planning 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.

Part II: Oracle Server Configuration

Chapter 4, "Managing Oracle Processes"	This chapter helps you identify different Oracle processes, such as dedicated server processes and multi-threaded server processes. Consult this chapter when configuring, modifying, tracking and managing processes.
Chapter 5, "Managing Control Files"	This chapter describes all aspects of managing control files (such as naming, creating, troubleshooting, and dropping control files).
Chapter 6, "Managing the Online Redo Log"	This chapter describes all aspects of managing the online redo log: planning, creating, renaming, dropping, or clearing online redo log files.
Chapter 7, "Managing Archived Redo Logs"	Consult this chapter for information about archive modes, tuning archiving, and viewing.
Chapter 8, "Managing Job Queues"	Consult this chapter before working with job queues. All aspects of submitting, removing, altering, and fixing job queues are described.

Part III: Database Storage

Chapter 9, "Managing Tablespaces"	This chapter provides guidelines to follow as you manage tablespaces, and describes how to create, manage, alter, drop and move data betweentablespaces.
Chapter 10, "Managing Datafiles"	This chapter provides guidelines to follow as you manage datafiles, and describes how to create, change, alter, rename and view information about datafiles.
Chapter 11, "Using the Database Resource Manager"	This chapter describes how to use the Database Resource Manager to allocate resources.
Chapter 12, "Guidelines for Managing Schema Objects"	Consult this chapter for descriptions of common tasks, such as setting storage parameters, deallocating space and managing space.
Chapter 13, "Managing Partitioned Tables and Indexes"	This chapter describes what a partitioned table (and index) is and how to create and manage it.
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 Views, Sequences and Synonyms"	This chapter describes all aspects of managing views, sequences and synonyms.
Chapter 16, "Managing Indexes"	Consult this chapter for general guidelines about indexes, including creating, altering, monitoring and dropping indexes.
Chapter 17, "Managing Clusters"	Consult this chapter for general guidelines to follow when creating, altering and dropping clusters.

Chapter 18, "Managing Hash Clusters"	Consult this chapter for general guidelines to follow when altering or dropping hash clusters.
Chapter 19, "Detecting and Repairing Data Block Corruption"	This chapter describes how to use the procedures in the DBMS_REPAIR package to detect and correct data block corruption.
Chapter 20, "General Management of Schema Objects"	This chapter covers more specific aspects of schema management than those identified in Chapter 12. Consult this chapter for information about table analysis, truncation of tables and clusters, database triggers, integrity constraints, object dependencies. You will also find a number of specific examples.
Chapter 21, "Managing Rollback Segments"	Consult this chapter for guidelines to follow when working with rollback segments.

Part IV: Database Security

Chapter 22, "Establishing Security Policies"	This chapter describes all aspects of database security, including system, data and user security policies, as well as specific tasks associated with password management.
Chapter 23, "Managing Users and Resources"	This chapter describes session and user licensing, user authentication, and provides specific examples of tasks associated with managing users and resources.
Chapter 24, "Managing User Privileges and Roles"	This chapter contains information about all aspects of managing user privileges and roles. Consult this chapter to find out how to grant and revoke privileges and roles.
Chapter 25, "Auditing Database Use"	This chapter describes how to create, manage and view audit information.

Conventions

This section explains the conventions used in this manual including the following:

- text
- syntax diagrams and notation
- code examples

Text

This section explains the conventions used within the text.

UPPERCASE Characters

Uppercase text is used to call attention to command keywords, object names, parameters, filenames, and so on.

For example, "If you create a private rollback segment, the name must be included in the ROLLBACK_SEGMENTS parameter of the parameter file."

Italicized Characters

Italicized words within text are book titles or emphasized words.

Syntax Diagrams and Notation

The syntax diagrams and notation in this manual show the syntax for SQL commands, functions, hints, and other elements. This section tells you how to read syntax diagrams and examples and write SQL statements based on them.

Keywords

Keywords are words that have special meanings in the SQL language. In the syntax diagrams in this manual, keywords appear in uppercase. You must use keywords in your SQL statements exactly as they appear in the syntax diagram, except that they can be either uppercase or lowercase. For example, you must use the CREATE keyword to begin your CREATE TABLE statements just as it appears in the CREATE TABLE syntax diagram.

Parameters

Parameters act as place holders in syntax diagrams. They appear in lowercase. Parameters are usually names of database objects, Oracle datatype names, or expressions. When you see a parameter in a syntax diagram, substitute an object or expression of the appropriate type in your SQL statement. For example, to write a

CREATE TABLE statement, use the name of the table you want to create, such as EMP, in place of the *table* parameter in the syntax diagram. (Note that parameter names appear in italics in the text.)

This list shows parameters that appear in the syntax diagrams in this manual and examples of the values you might substitute for them in your statements:

Parameter	Description	Examples
table	The substitution value must be the name of an object of the type specified by the parameter.	emp
'text'	The substitution value must be a character literal in single quotes.	'Employee Records'
condition	The substitution value must be a condition that evaluates to TRUE or FALSE.	ename > 'A'
date *d*	The substitution value must be a date constant or an expression of DATE datatype.	TO_DATE ('01-Jan-1996', DD-MON-YYYY')
expr	The substitution value can be an expression of any datatype.	sal + 1000
integer	The substitution value must be an integer.	72
rowid	The substitution value must be an expression of datatype ROWID.	00000462.0001.0001
subquery	The substitution value must be a SELECT statement contained in another SQL statement.	SELECT ename FROM emp
statement_name *block_name*	The substitution value must be an identifier for a SQL statement or PL/SQL block.	s1 b1

Code Examples

SQL and SQL*Plus commands and statements are separated from the text of paragraphs in a monospaced font as follows:

```
INSERT INTO emp (empno, ename) VALUES (1000, 'JFEE);
```

```
ALTER TABLESPACE users ADD DATAFILE 'users2.ora' SIZE 50K;
```

Example statements can include punctuation, such as commas or quotation marks. All punctuation in example statements is required. All example statements terminate with a semicolon (;). Depending on the application, a semicolon or other terminator may or may not be required to end a statement.

Uppercase words in example statements indicate the keywords within Oracle SQL. When you issue statements, however, keywords are not case sensitive.

Lowercase words in example statements indicate words supplied only for the context of the example. For example, lowercase words may indicate the name of a table, column, or file.

Part I

Basic Database Administration

1

The Oracle Database Administrator

This chapter describes the responsibilities of the person who administers the Oracle server, the database administrator.

The following topics are included:

- Types of Oracle Users
- Database Administrator Security and Privileges
- Database Administrator Authentication
- Password File Administration
- Database Administrator Utilities
- Priorities of a Database Administrator
- Identifying Oracle Software Releases

Types of Oracle Users

At your site, the types of users and their responsibilities may vary. For example, at a large site the duties of a database administrator might be divided among several people.

This section includes the following topics:

Database Administrators

Because an Oracle database system can be quite large and have many users, someone or some group of people must manage this system. The *database administrator* (DBA) is this manager. Every database requires at least one person to perform administrative duties.

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

- installing and upgrading the Oracle 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 your Oracle license agreement
- 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 Corporation for technical support

Security Officers

In some cases, a database might also have one or more security officers. A *security officer* is primarily concerned with enrolling users, controlling and monitoring user access to the database, and maintaining system security. You might not be responsible for these duties if your site has a separate security officer.

Application Developers

An *application developer* designs and implements database applications An application developer's 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 the above information to a database administrator
- tuning the application during development
- establishing an application's security measures during development

Application Administrators

An Oracle site might also have one or more application administrators. An *application administrator* is responsible for the administration needs of a particular application.

Database Users

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

- entering, modifying, and deleting data, where permitted

- generating reports of data

Network Administrators

At some sites there may be one or more network administrators. Network administrators may be responsible for administering Oracle networking products, such as Net8.

See Also: "Network Administration" in *Oracle8i Distributed Database Systems*

Database Administrator Security and Privileges

To accomplish administrative tasks in Oracle, you need extra privileges both 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 includes the following topics:

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

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 that executes Oracle, you might need an operating system account or ID to gain access to the operating system. If so, your operating system account might require more operating system privileges or access rights than many database users require (for example, to perform Oracle software installation). Although you do not need the Oracle files to be stored in your account, you should have access to them.

In addition, Enterprise Manager requires that your operating system account or ID be distinguished in some way to allow you to use *operating system privileged* Enterprise Manager commands.

See Also: The method of distinguishing a database administrator's account is operating system specific. See your operating system-specific Oracle documentation for information.

Database Administrator Usernames

Two user accounts are automatically created with the database and granted the DBA role. These two user accounts are:

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

These two usernames are described in the following sections.

> **Note:** To prevent inappropriate access to the data dictionary tables, you must change the passwords for the SYS and SYSTEM usernames immediately after creating an Oracle database.

You will probably want to create at least one additional administrator username to use when performing daily administrative tasks.

SYS

When any database is created, the user SYS, identified by the password CHANGE_ON_INSTALL, is automatically created and granted the DBA role.

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

Most database users should never be able to connect using the SYS account. You can connect to the database using this account but should do so only when instructed by Oracle personnel or documentation.

SYSTEM

When a database is created, the user SYSTEM, identified by the password MANAGER, is also automatically created and granted all system privileges for the database.

The SYSTEM username creates additional tables and views that display administrative information, and internal tables and views used by Oracle tools. Never create in the SYSTEM schema tables of interest to individual users.

The DBA Role

A predefined role, named "DBA", is automatically created with every Oracle database. This role contains all database system privileges. Therefore, it is very powerful and should be granted only to fully functional database administrators.

Database Administrator Authentication

Database administrators must often perform special operations such as shutting down or starting up a database. Because these operations should not be performed by normal database users, the database administrator usernames need a more secure authentication scheme.

This section includes the following topics:

- Selecting an Authentication Method
- Using Operating System Authentication
- OSOPER and OSDBA
- Using an Authentication Password File

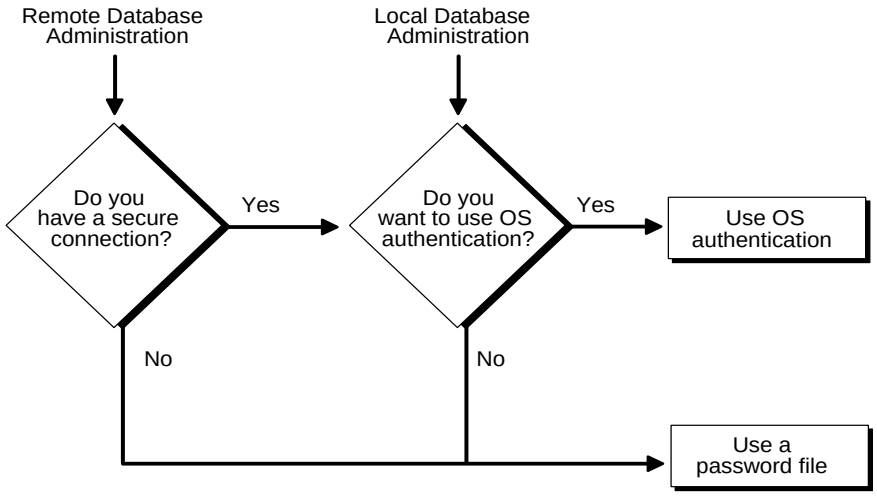
Selecting an Authentication Method

The following methods for authenticating database administrators replace the CONNECT INTERNAL syntax provided with earlier versions of Oracle:

- operating system authentication
- password files

Depending on whether you wish to administer your database locally on the same machine where the database resides or to administer many different databases from a single remote client, you can choose between operating system authentication or password files to authenticate database administrators. Figure 1–1 illustrates the choices you have for database administrator authentication schemes.

Figure 1–1 Database Administrator Authentication Methods



On most operating systems, OS authentication for database administrators involves placing the OS username of the database administrator in a special group (on UNIX systems, this is the DBA group) or giving that OS username a special process right.

The database uses password files to keep track of database usernames that have been granted administrator privileges.

See Also: "User Authentication" in *Oracle8i Concepts.*

Using Operating System Authentication

If you choose, you can have your operating system authenticate users performing database administration operations.

1. Set up the user to be authenticated by the operating system.
2. Make sure that the initialization parameter, REMOTE_LOGIN_PASSWORDFILE, is set to NONE, which is the default value for this parameter.
3. Authenticated users should now be able to connect to a local database, or to connect to a remote database over a secure connection, by typing one of the following commands:

```
CONNECT / AS SYSOPER
```

```
CONNECT / AS SYSDBA
```

If you successfully connect as INTERNAL using an earlier release of Oracle, you should be able to continue to connect successfully using the new syntax shown in Step 3.

Note: To connect as SYSOPER or SYSDBA using OS authentication you do not need the SYSOPER or SYSDBA system privileges. Instead, the server verifies that you have been granted the appropriate OSDBA or OSOPER roles at the operating system level.

OSOPER and OSDBA

Two special operating system roles control database administrator logins when using operating system authentication: OSOPER and OSDBA.

OSOPER	Permits the user to perform STARTUP, SHUTDOWN, ALTER DATABASE OPEN/MOUNT, ALTER DATABASE BACKUP, ARCHIVE LOG, and RECOVER, and includes the RESTRICTED SESSION privilege.
OSDBA	Contains all system privileges with ADMIN OPTION, and the OSOPER role; permits CREATE DATABASE and time-based recovery.

OSOPER and OSDBA can have different names and functionality, depending on your operating system.

The OSOPER and OSDBA roles can only be granted to a user through the operating system. They cannot be granted through a GRANT statement, nor can they be revoked or dropped. When a user logs on with administrator privileges and REMOTE_LOGIN_PASSWORDFILE is set to NONE, Oracle communicates with the operating system and attempts to enable first OSDBA and then, if unsuccessful, OSOPER. If both attempts fail, the connection fails. How you grant these privileges through the operating system is operating system specific.

If you are performing remote database administration, you should consult your Net8 documentation to determine if you are using a secure connection. Most popular connection protocols, such as TCP/IP and DECnet, are not secure, regardless of which version of Net8 you are using.

See Also: For information about OS authentication of database administrators, see your operating system-specific Oracle documentation.

Using an Authentication Password File

If you have determined that you need to use a password file to authenticate users performing database administration, you must complete the steps outlined below. Each of these steps is explained in more detail in the following sections of this chapter.

1. Create the password file using the ORAPWD utility.

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

2. Set the REMOTE_LOGIN_PASSWORDFILE initialization parameter to EXCLUSIVE.

3. Add users to the password file by using SQL to grant the appropriate privileges to each user who needs to perform database administration, as shown in the following examples.

   ```
   GRANT SYSDBA TO scott;
   GRANT SYSOPER TO scott;
   ```

   The privilege SYSDBA permits the user to perform the same operations as OSDBA. Likewise, the privilege SYSOPER permits the user to perform the same operations as OSOPER.

4. Privileged users should now be able to connect to the database by using a command similar to the one shown below.

   ```
   CONNECT scott/tiger@acct.hq.com AS SYSDBA
   ```

Password File Administration

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

This section includes the following topics:

- Using ORAPWD
- Setting REMOTE_LOGIN_ PASSWORDFILE
- Adding Users to a Password File
- Connecting with Administrator Privileges
- Maintaining a Password File

See Also: See your operating system-specific Oracle documentation for information on using the installer utility to install the password file.

Using ORAPWD

When you invoke the 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 and INTERNAL (mand),
    entries - maximum number of distinct DBAs and OPERs (opt),
    There are no spaces around the equal-to (=) character.
```

For example, the following command creates a password file named ACCT.PWD that allows up to 30 privileged users with different passwords. The file is initially created with the password SECRET for users connecting as SYSOPER or SYSDBA:

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

Following are descriptions of the parameters in the ORAPWD utility.

FILE

This parameter sets the name of the password file being created. You must specify the full pathname for the file. The contents of this file are encrypted, and the file is not user-readable. This parameter is mandatory.

The types of file names allowed for the password file are operating system specific. Some platforms require the password file to be a specific format and located in a specific directory. Other platforms allow the use of environment variables to specify the name and location of the password file. See your operating system-specific Oracle documentation for the names and locations allowed on your platform.

If you are running multiple instances of Oracle using the Oracle Parallel Server, the environment variable for each instance should point to the same password file.

> **WARNING: It is critically important to the security of your system that you protect your password file and 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 SYSOPER and SYSDBA. If you issue the ALTER USER command to change the password after connecting to the database, both the password stored in the data dictionary and the password stored in the password file are updated. The INTERNAL user is supported for backwards compatibility only. This parameter is mandatory.

ENTRIES
This parameter sets the maximum number of entries allowed in the password file. This corresponds to the maximum number of distinct users allowed to connect to the database as SYSDBA or SYSOPER. Entries can be reused as users are added to and removed from the password file. This parameter is required if you ever want this password file to be EXCLUSIVE.

> **WARNING: If you ever need to exceed this limit, you must create a new password file. It is safest to select a number larger than you think you will ever need.**

See Also: Consult your operating system-specific Oracle documentation for the exact name of the password file or for the name of the environment variable used to specify this name for your operating system.

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 described below.

> **Note:** To start up an instance or database, you must use Enterprise Manager. You must specify a database name and a parameter file to initialize the instance settings. You may specify a fully-qualified remote database name using Net8. However, the initialization parameter file and any associated files, such as a configuration file, must exist on the client machine. That is, the parameter file must be on the machine where you are running Enterprise Manager.

NONE
Setting this parameter to NONE causes Oracle to behave as if the password file does not exist. That is, no privileged connections are allowed over non-secure 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 SYSOPER and SYSDBA. Using an EXCLUSIVE password file allows you to 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. However, the only users recognized by a SHARED password file are SYSDBA and SYSOPER; 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 file 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 is 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, you receive an error message if you attempt to grant these privileges.

A user's name only remains in the password file while that user has at least one of these two privileges. When you revoke the last of these privileges from a user, that user is removed from the password file.

To Create a Password File and Add New Users to It

1. Follow the instructions for creating a password file.
2. Set the REMOTE_LOGIN_PASSWORDFILE initialization parameter to EXCLUSIVE.

3. Connect with SYSDBA privileges as shown in the following example:

```
CONNECT SYS/change_on_install 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 SYSOPER or SYSDBA privileges to yourself and other users as appropriate.
6. These users are now added to the password file and can connect to the database as SYSOPER or SYSDBA with a username and password (instead of using SYS). The use of a password file does not prevent OS authenticated users from connecting if they meet the criteria for OS authentication.

Granting and Revoking SYSOPER and SYSDBA Privileges

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

```
GRANT SYSDBA TO scott;
```

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

```
REVOKE SYSDBA FROM scott;
```

Because SYSDBA and SYSOPER are the most powerful database privileges, the ADMIN OPTION is not used. Only users currently connected as SYSDBA (or INTERNAL) can grant SYSDBA or SYSOPER system privileges to another user. This is also true of REVOKE. These privileges cannot be granted to roles, since roles are only available after database startup. Do not confuse the SYSDBA and SYSOPER database privileges with operating system roles, which are a completely independent feature.

See Also: For more information about system privileges, see Chapter 24, "Managing User Privileges and Roles".

Listing Password File Members

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

USERNAME
The name of the user that is recognized by the password file.

SYSDBA
If the value of this column is TRUE, the user can log on with SYSDBA system privileges.

SYSOPER
If the value of this column is TRUE, the user can log on with SYSOPER system privileges.

Connecting with Administrator Privileges

When you connect with SYSOPER or SYSDBA privileges using a username and password, you are connecting with a default schema of SYS, not the schema that is generally associated with your username.

Connecting with Administrator Privileges: Example

For example, assume user SCOTT has issued the following commands:

```
CONNECT scott/tiger
CREATE TABLE scott_test(name VARCHAR2(20));
```

Later, when SCOTT issues these commands:

```
CONNECT scott/tiger AS SYSDBA
SELECT * FROM scott_test;
```

He receives an error that SCOTT_TEST does not exist. That is because SCOTT now references the SYS schema by default, whereas the table was created in the SCOTT schema.

Non-Secure Remote Connections

To connect to Oracle as a privileged user over a non-secure connection, you must meet the following conditions:

- The server to which you are connecting must have a password file.
- You must be granted the SYSOPER or SYSDBA system privilege.
- You must connect using a username and password.

Local and Secure Remote Connections

To connect to Oracle as a privileged user over a local or a secure remote connection, you must meet either of the following sets of conditions:

- You can connect using a password file, provided that you meet the criteria outlined for non-secure connections in the previous bulleted list.
- If the server is not using a password file, or you have not been granted SYSOPER or SYSDBA privileges and are therefore not in the password file, your operating system name must be authenticated for a privileged connection by the operating system. This form of authentication is operating system specific.

Consult your operating system-specific Oracle documentation for details on operating system authentication.

See Also: "Password File Administration" on page 1-9.

Maintaining a Password File

This section describes how to expand, relocate, and remove the password file, as well as how to 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 re-grant the privileges to the users.

To Replace a Password File

1. Note which users 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 1-10. Be sure to set the ENTRIES parameter to a sufficiently large number.
5. Follow the instructions in "Adding Users to a Password File" on page 1-12.

Relocating the Password File

After you have created the password file, you can relocate it as you choose. After relocating the password file, you must reset the appropriate environment variables to the new pathname. If your operating system uses a predefined pathname, you cannot change the password file location.

Removing a Password File

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

> **WARNING: 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 timestamp 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 parameter REMOTE_LOGIN_PASSWORDFILE. When you start up an instance, Oracle retrieves the value of this parameter from the initialization parameter file stored on your client machine. When you mount the database, Oracle compares the value of this parameter to the value stored in the password file. If these values do not match, the value stored in the file is overwritten.

> **WARNING: You should use caution to ensure that an EXCLUSIVE password file is not accidentally changed to SHARED. If you plan to allow instance start up 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.**

Database Administrator Utilities

Several utilities are available to help you maintain and control the Oracle server.

The following topics are included in this section:

- SQL*Loader
- Export and Import

SQL*Loader

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

See Also: *Oracle8i Utilities*

Export and Import

The Export and Import utilities allow you to move existing data in Oracle format to and from Oracle databases. For example, export files can archive database data, or move data among different Oracle databases that run on the same or different operating systems.

See Also: *Oracle8i Utilities*

Priorities of a Database Administrator

In general, you must perform a series of steps to get the database system up and running, and then maintain it.

The following steps are required to configure an Oracle server and database on any type of computer system. The following sections include details about each step.

To Configure an Oracle Server

- Step 1: Install the Oracle Software
- Step 2: Evaluate the Database Server Hardware
- Step 3: Plan the Database
- Step 4: Create and Open the Database
- Step 5: Implement the Database Design

- Step 6: Back Up the Database
- Step 7: Enroll System Users
- Step 8: Tune Database Performance

> **Note:** If migrating to a new release, back up your existing production database before installation. For more information on preserving your existing production database, see *Oracle8i Migration*.

Step 1: Install the Oracle Software

As the database administrator, you must install the Oracle 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 and the database tools and applications are executed on remote machines; in this case, you must also install the Oracle Net8 drivers necessary to connect the remote machines to the computer that executes Oracle.

See Also: For more information, see "Identifying Oracle Software Releases" on page 1-21.

For specific requirements and instructions for installation, see your operating system-specific Oracle documentation and your installation guides for your front-end tools and Net8 drivers.

Step 2: Evaluate the Database Server Hardware

After installation, evaluate how Oracle and its applications can best use the available computer resources. This evaluation should reveal the following information:

- how many disk drives are available to Oracle and its databases
- how many, if any, dedicated tape drives are available to Oracle and its databases
- how much memory is available to the instances of Oracle you will run (see your system's configuration documentation)

Step 3: Plan the Database

As the database administrator, you must plan:

- the database's logical storage structure
- 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 data files will make up the tablespace, where the data files will be physically stored (on which disk drives), and what type of information will be stored in each tablespace. When planning the database's overall logical storage structure, take into account the effects that this structure will have when the database is actually created and running. Such considerations include how the database's logical storage structure will affect the following items:

- the performance of the computer executing Oracle
- 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's objects and the storage characteristics for each of these objects. By planning relationships between objects and the physical storage of each object before creating it, you can directly impact 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 can dramatically affect application performance.

During the above planning phases, also plan a backup strategy for the database. After developing this strategy, you might find that you want to alter the database's planned logical storage structure or database design 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, refer to accepted industry-standard books that explain these studies.

See Also: See *Chapters 9 through 21* for specific information on creating logical storage structures, objects, and integrity constraints for your database.

Step 4: Create and Open the Database

Once you have finalized the database design, you can create the database and open it for normal use. Depending on your operating system, a database may already

have been created during the installation procedure for Oracle. If so, all you need to do is start an instance and mount and open the initial database.

To determine if your operating system creates an initial database during the installation of Oracle, check your installation or user's guide. If no database is created during installation or you want to create an additional database, see Chapter 2 of this book for this procedure.

See Also: See Chapter 3 for database and instance startup and shutdown procedures.

Step 5: Implement the Database Design

Once you have created and started the database, you can create the database's planned logical structure by creating all necessary rollback segments and tablespaces. Once this is built, you can create the objects for your database.

See Also: See Chapters 9 through 21 for instructions on creating logical storage structures and objects for your database.

Step 6: Back Up the Database

After you have created the database structure, carry out the planned backup strategy for your database by creating any additional redo log files, taking the first full database backup (online or offline), and scheduling future database backups at regular intervals.

See Also: See the *Oracle8i Backup and Recovery Guide* for instructions on customizing your backup operations and performing recovery procedures.

Step 7: Enroll System Users

Once you have backed up the database structure, you can begin to enroll the users of the database in accordance with your Oracle license agreement, create roles for these users, and grant appropriate roles to them.

See Also: See Chapters 22 through 24 for the procedures to create user accounts and roles, and information on complying with your license agreement.

Step 8: Tune Database Performance

Optimizing the database system's performance is one of your ongoing responsibilities.

See Also: *Oracle8i Tuning* for information about tuning your database and applications.

Identifying Oracle Software Releases

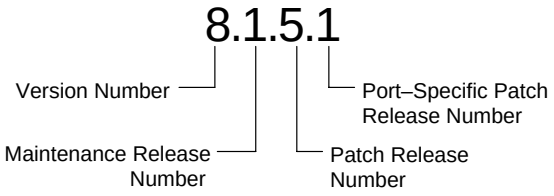
Because Oracle products are always undergoing development and change, several releases of the products can be in use at any one time. To identify a software product fully, as many as five numbers may be required.

This section includes the following topics:

Release Number Format

An Oracle Server distribution tape might be labeled "Release 8.1.5.1." The following sections translate this number.

Figure 1–2 Example of an Oracle Release Number



Version Number

The version number, such as **8**, is the most general identifier. A *version* is a major new edition of the software, which usually contains significant new functionality.

Maintenance Release Number

The maintenance release number signifies different releases of the general version, starting with 0, as in version 8.**0**. The maintenance release number increases when bug fixes or new features to existing programs become available; in the example here, 8.**1** indicates the first maintenance release of the version 8.

Patch Release Number

The patch release number identifies a specific level of the object code, such as 8.1.**5**. A patch release contains fixes for serious bugs that cannot wait until the next maintenance release. The first distribution of a maintenance release always has a patch number of 0.

Port-Specific Patch Release Number

A fourth number (and sometimes a fifth number) can be used to identify a particular emergency patch release of a software product on that operating system, such as 8.1.5.**1.** or 8.1.5.**1.3.** An emergency patch is not usually intended for wide distribution; it usually fixes or works around a particular, critical problem.

Examples of Release Numbers

The following examples show possible release numbers for Oracle8*i*:

8.0.0	the first distribution of Oracle8*i*
8.1.0	the first maintenance release of Oracle8*i*
8.2.0	the second maintenance release (the third release in all) of Oracle8*i*
8.2.2	the second patch release after the second maintenance release

Versions of Other Oracle Software

As Oracle Corporation introduces new products and enhances existing ones, the version numbers of the individual products increment independently. Thus, you might have an Oracle server Release 8.1.5.1 system working with Oracle Forms Version 4.0.3, SQL*Plus Version 3.1.9, and Pro*FORTRAN Version 1.5.2. (These numbers are used only for illustration.)

Checking Your Current Release Number

To see which release of Oracle and its components you are using, query the data dictionary view PRODUCT_COMPONENT_VERSION, as shown below (This information is useful if you need to call Oracle Support.):

```
SELECT * FROM product_component_version;

PRODUCT                  VERSION              STATUS
-------------------      -----------------    ----------
CORE                         3.4.1.0.0         Production
```

```
NLSRTL                         3.1.3.0.0         Production
Oracle8i Server                8.1.4.0.0         Beta Release
PL/SQL                         2.2.1.0.0         Beta
TNS for SunOS:                 2.1.4.0.0         Production
5 rows selected
```

2

Creating an Oracle Database

This chapter lists the steps necessary to create an Oracle database, and includes the following topics:

- Considerations Before Creating a Database
- Creating an Oracle Database
- Parameters
- Considerations After Creating a Database
- Initial Tuning Guidelines

Considerations Before Creating a Database

This section includes the following topics:

Database creation prepares several operating system files so they can 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. Creating a database can also erase information in an existing database and create a new database with the same name and physical structure.

Creating a database includes the following operations:

- creating new datafiles or erasing data that existed in previous datafiles
- creating information structures that Oracle requires to access and use the database (the data dictionary)
- creating and initializing the control files and redo log files for the database

Consider the following issues before you create a database:

- Plan your database tables and indexes, and estimate how much space they will require.
- Plan how to protect your new database, including the configuration of its online and archived redo log (and how much space it will require), and a backup strategy.
- Select the database character set. You must specify the database character set when you create the database. After the database is created, you cannot change the character set choices without re-creating the database. Hence, it is important that you carefully consider which character set(s) to use. All character data, including data in the data dictionary, is stored in the database character set. If users access the database using a different character set, the database character set should be the same as, or a superset of, all character sets they use.

Also become familiar with the principles and options of starting up and shutting down an instance, mounting and opening a database, and using parameter files.

See Also: The *Oracle8i National Language Support Guide.*

For information about tables, indexes, and space management, see Chapters 12 through 19.

For information about the online and archive redo logs, and database backup and recovery see Chapter 6, "Managing the Online Redo Log" and Chapter 7, "Managing Archived Redo Logs".

Creation Prerequisites

To create a new database, you must have the following:

- the operating system privileges associated with a fully operational database administrator
- sufficient memory to start the Oracle instance
- sufficient disk storage space for the planned database on the computer that executes Oracle

Using an Initial Database

Depending on your operating system, a database might have been created automatically as part of the installation procedure for Oracle. You can use this initial database and customize it to meet your information management requirements, or discard it and create one or more new databases to replace it.

Migrating an Older Version of the Database

If you are using a previous release of Oracle, database creation is required only if you want an entirely new database. Otherwise, you can migrate your existing Oracle databases managed by a previous version of Oracle and use them with the new version of the Oracle software.

See Also: *Oracle8i Migration* manual for information about migrating an existing database.

For more information about migrating an existing database, see your operating system-specific Oracle documentation.

Creating an Oracle Database

This section includes the following topics:

- Steps for Creating an Oracle Database
- Creating a Database: Example
- Troubleshooting Database Creation

- Dropping a Database

Steps for Creating an Oracle Database

These steps, which describe how to create an Oracle database, should be followed in the order presented.

To Create a New Database and Make It Available for System Use

1. Back up any existing databases.
2. Create parameter files.
3. Edit new parameter files.
4. Check the instance identifier for your system.
5. Start SQL*Plus and connect to Oracle as SYSDBA.
6. Start an instance.
7. Create the database.
8. Back up the database.

See Also: These steps provide general information about database creation on all operating systems. See your operating system-specific Oracle documentation for information about creating databases on your platform.

Step 1: Back up any existing databases. Oracle Corporation strongly recommends that you make complete backups of all existing databases before creating a new database, in case database creation accidentally affects some existing files. Backup should include parameter files, datafiles, redo log files, and control files.

Step 2: Create parameter files. The instance (System Global Area and background processes) for any Oracle database is started using a parameter file.

Each database on your system should have at least one customized parameter file that corresponds only to that database. Do not use the same file for several databases.

To create a parameter file for the database you are about to make, use your operating system to make a copy of the parameter file that Oracle provided on the distribution media. Give this copy a new filename. You can then edit and customize this new file for the new database.

See Also: For more information about copying the parameter file, see your operating system-specific Oracle documentation.

> **Note:** In distributed processing environments, Enterprise Manager is often executed from a client machine of the network. If a client machine is being used to execute Enterprise Manager and create a new database, you need to copy the new parameter file (currently located on the computer executing Oracle) to your client workstation. This procedure is operating system dependent. For more information about copying files among the computers of your network, see your operating system-specific Oracle documentation.

Step 3: Edit new parameter files. To create a new database, inspect and edit the following parameters of the new parameter file:

Parameter	Described
DB_NAME	on page 2-9
DB_DOMAIN	on page 2-9
CONTROL_FILES	on page 2-10
DB_BLOCK_SIZE	on page 2-11
DB_BLOCK_BUFFERS	on page 2-11
PROCESSES	on page 2-12
ROLLBACK_SEGMENTS	on page 2-12

You should also edit the appropriate license parameter(s):

Parameter	Described
LICENSE_MAX_SESSIONS	on page 2-13
LICENSE_SESSION_WARNING	on page 2-13
LICENSE_MAX_USERS	on page 2-13

Step 4: Check the instance identifier for your system. If you have other databases, check the Oracle instances identifier. The Oracle instance identifier should match the name of the database (the value of DB_NAME) to avoid confusion with other Oracle instances that are running concurrently on your system.

See your operating system-specific Oracle documentation for more information.

Step 5: Start SQL*Plus and connect to Oracle as SYSDBA. Connect to the database as SYSDBA.

```
$ SQLPLUS /nolog
connect username/password as sysdba
```

Step 6: Start an instance. You can start an instance without mounting a database; typically, you do so only during database creation. Use the STARTUP command with the NOMOUNT option:

```
STARTUP NOMOUNT;
```

At this point, there is no database. Only an SGA and background processes are started in preparation for the creation of a new database.

Step 7: Create the database. To create the new database, use the SQL CREATE DATABASE statement, optionally setting parameters within the statement to name the database, establish maximum numbers of files, name the files and set their sizes, and so on.

When you execute a CREATE DATABASE statement, Oracle performs the following operations:

- creates the datafiles for the database
- creates the control files for the database
- creates the redo log files for the database
- creates the SYSTEM tablespace and the SYSTEM rollback segment
- creates the data dictionary
- creates the users SYS and SYSTEM
- specifies the character set that stores data in the database
- mounts and opens the database for use

> **WARNING: Make sure that the datafile and redo log file names that you specify do not conflict with files of another database.**

See Also: You can also create a database with a locally managed SYSTEM tablespace; for more information, see "Creating a Database with a Locally Managed SYSTEM Tablespace" on page 9-5.

Step 8: Back up the database. You should make 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.

See Also: The *Oracle8i Backup and Recovery Guide*.

For more information about parameter files see "Using Parameter Files" on page 3-13.

For information about the CREATE DATABASE statement, character sets, and database creation see the *Oracle8i SQL Reference*.

Creating a Database: Example

The following statement is an example of a CREATE DATABASE statement:

```
CREATE DATABASE test
     DATAFILE 'test_system' SIZE 10M
     LOGFILE GROUP 1 ('test_log1a', 'test_log1b') SIZE 500K,
     GROUP 2 ('test_log2a', 'test_log2b') SIZE 500K;
```

The values of the MAXLOGFILES, MAXLOGMEMBERS, MAXDATAFILES, MAXLOGHISTORY, and MAXINSTANCES options in this example assume the default values, which are operating system-dependent. The database is mounted in the default modes NOARCHIVELOG and EXCLUSIVE and then opened.

The items and information in the example statement above result in creating a database with the following characteristics:

- The new database is named TEST.
- The SYSTEM tablespace of the new database is comprised of one 10 MB datafile named TEST_SYSTEM.
- The new database has two online redo log groups, each containing two 500 KB members.

- The new database does not overwrite any existing control files specified in the parameter file.

> **Note:** You can set several limits during database creation. Some of these limits are also subject to superseding limits of the operating system and can affect each other. For example, if you set MAXDATAFILES, Oracle allocates enough space in the control file to store MAXDATAFILES filenames, even if the database has only one datafile initially; 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.

See Also: For more information about setting limits during database creation, see the *Oracle8i SQL Reference*.

See your operating system-specific Oracle documentation for information about operating system limits.

Troubleshooting Database Creation

If for any reason database creation fails, shut down the instance and delete any files created by the CREATE DATABASE statement before you attempt to create it once again.

After correcting the error that caused the failure of the database creation, return to "Creating a Database: Example".

Dropping a Database

To drop a database, remove its datafiles, redo log files, and all other associated files (control files, parameter files, archived log files).

To view the names of the database's datafiles and redo log files, query the data dictionary views V$DATAFILE and V$LOGFILE.

See Also: For more information about these views, see the *Oracle8i Reference*.

Parameters

As described in Step 3 of the section "Creating an Oracle Database", Oracle suggests you alter a minimum set of parameters. These parameters are described in the following sections:

- DB_NAME and DB_DOMAIN
- CONTROL_FILES
- DB_BLOCK_SIZE
- PROCESSES
- ROLLBACK_SEGMENTS
- License Parameters
- DB_BLOCK_BUFFERS
- LICENSE_MAX_SESSIONS and LICENSE_SESSIONS _WARNING
- LICENSE_MAX_USERS

DB_NAME and DB_DOMAIN

A database's *global database name* (name and location within a network structure) is created by setting both the DB_NAME and DB_DOMAIN parameters before database creation. After creation, the database's name cannot be easily changed. The DB_NAME parameter determines the local name component of the database's name, while the DB_DOMAIN parameter indicates the domain (logical location) within a network structure. The combination of the settings for these two parameters should 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
```

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 (of the parameter file) and the database name in the control file are not the same, the database does not start.

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, pay special attention to this initialization parameter before database creation.

See Also: For more information about distributed databases, see *Oracle8i Distributed Database Systems*.

CONTROL_FILES

Include the CONTROL_FILES parameter in your new parameter file and set its value to a list of control filenames to use for the new database. If you want Oracle to create new operating system files when creating your database's control files, make sure that the filenames listed in the CONTROL_FILES parameter do not match any filenames that currently exist on your system. If you want Oracle to reuse or overwrite existing files when creating your database's control files, make sure that the filenames listed in the CONTROL_FILES parameter match the filenames that currently exist.

> **WARNING: Use extreme caution when setting this option. If you inadvertently specify a file that you did not intend and execute the CREATE DATABASE statement, the previous contents of that file will be overwritten.**

If no filenames are listed for the CONTROL_FILES parameter, Oracle uses a default filename.

Oracle Corporation strongly recommends you use at least two control files stored on separate physical disk drives for each database. Therefore, when specifying the CONTROL_FILES parameter of the new parameter file, follow these guidelines:

- List at least two filenames for the CONTROL_FILES parameter.
- Place each control file on a separate physical disk drives by fully specifying filenames that refer to different disk drives for each filename.

> **Note:** The file specification for control files is operating system-dependent. Regardless of your operating system, *always* fully specify filenames for your control files.

When you execute the CREATE DATABASE statement (in Step 7), the control files listed in the CONTROL_FILES parameter of the parameter file will be created.

See Also: The default filename for the CONTROL_FILES parameter is operating system-dependent. See your operating system-specific Oracle documentation for details.

DB_BLOCK_SIZE

The default data block size for every Oracle server is operating system-specific. The Oracle data block size is typically either 2K or 4K. Generally, the default data block size is adequate. In some cases, however, a larger data block size provides greater efficiency in disk and memory I/O (access and storage of data). Such cases include:

- Oracle 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 uses a small operating system block size. For example, if the operating system block size is 1K and the data block size matches this, Oracle 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.

Each database's block size is set during database creation by the initialization parameter DB_BLOCK_SIZE. The block size *cannot* be changed after database creation except by re-creating the database. If a database's block size is different from the operating system block size, make the database block size a multiple of the operating system's block size.

For example, if your operating system's block size is 2K (2048 bytes), the following setting for the DB_BLOCK_SIZE initialization parameter would be valid:

```
DB_BLOCK_SIZE=4096
```

DB_BLOCK_SIZE also determines the size of the database buffers in the buffer cache of the System Global Area (SGA).

See Also: For details about your default block size, see your operating system-specific Oracle documentation.

DB_BLOCK_BUFFERS

This parameter determines the number of buffers in the buffer cache in the System Global Area (SGA). The number of buffers affects the performance of the cache. Larger cache sizes reduce the number of disk writes of modified data. However, a

large cache may take up too much memory and induce memory paging or swapping.

Estimate the number of data blocks that your application accesses most frequently, including tables, indexes, and rollback segments. This estimate is a rough approximation of the minimum number of buffers the cache should have. Typically, 1000 to 2000 is a practical minimum for the number of buffers.

See Also: For more information about tuning the buffer cache, see *Oracle8i Tuning*.

PROCESSES

This parameter determines the maximum number of operating system processes that can be connected to Oracle concurrently. The value of this parameter must include 5 for the background processes and 1 for each user process. For example, if you plan to have 50 concurrent users, set this parameter to at least 55.

ROLLBACK_SEGMENTS

This parameter is a list of the rollback segments an Oracle instance acquires at database startup. List your rollback segments as the value of this parameter.

Attention: After installation, you must create at least one rollback segment in the SYSTEM tablespace in addition to the SYSTEM rollback segment before you can create any schema objects.

See Also: For more information about how many rollback segments you need, see *Oracle8i Tuning*.

License Parameters

Oracle helps you ensure that your site complies with its Oracle license agreement. If your site is licensed by concurrent usage, you can track and limit the number of sessions concurrently connected to an instance. If your site is licensed by named users, you can limit the number of named users created in a database. To use this facility, you need to know which type of licensing agreement your site has and what the maximum number of sessions or named users is. Your site might use either type of licensing (session licensing or named user licensing), but not both.

See Also: For more information about managing licensing, see "Session and User Licensing" on page 23-2.

LICENSE_MAX_SESSIONS and LICENSE_SESSIONS _WARNING

You can set a limit on the number of concurrent sessions that can connect to a database on the specified computer. To set the maximum number of concurrent sessions for an instance, set the parameter LICENSE_MAX_SESSIONS in the parameter file that starts the instance, as shown in the following example:

```
LICENSE_MAX_SESSIONS = 80
```

In addition to setting a maximum number of sessions, you can set a warning limit on the number of concurrent sessions. Once this limit is reached, additional users can continue to connect (up to the maximum limit), but Oracle sends a warning for each connecting user. To set the warning limit for an instance, set the parameter LICENSE_SESSIONS_WARNING. Set the warning limit to a value lower than LICENSE_MAX_SESSIONS.

For instances running with the Parallel Server, each instance can have its own concurrent usage limit and warning limit. However, the sum of the instances' limits must not exceed the site's session license.

See Also: For more information about setting these limits when using the Parallel Server, see *Oracle8i Parallel Server Concepts and Administration*.

LICENSE_MAX_USERS

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 parameter in the database's parameter file, as shown in the following example:

```
LICENSE_MAX_USERS = 200
```

For instances running with the Parallel Server, all instances connected to the same database should have the same named user limit.

See Also: For more information about setting this limit when using the Parallel Server see *Oracle8i Parallel Server Concepts and Administration.*

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. If more than one database exists in your database system, specify the parameter file to use with any subsequent database startup.

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. See your operating system-specific Oracle documentation for the additional products. Usually, command files are provided to create and load these tables into the database's data dictionary.

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

A newly created database has only two users, SYS and SYSTEM. The passwords for these two usernames should be changed soon after the database is created.

See Also: For more information about the users SYS and SYSTEM see "Database Administrator Usernames" on page 1-5.

For information about changing a user's password see "Altering Users" on page 23-15.

Initial Tuning Guidelines

You can make a few significant tuning alterations to Oracle immediately following installation. By following these instructions, you can reduce the need to tune Oracle when it is running. This section gives recommendations for the following installation issues:

- Allocating Rollback Segments
- Choosing the Number of DB_BLOCK_LRU_LATCHES
- Distributing I/O

Allocating Rollback Segments

Proper allocation of rollback segments makes for optimal database performance. The size and number of rollback segments required for optimal performance depends on your application. *Oracle8i Tuning* contains some general guidelines for choosing how many rollback segments to allocate based on the number of concurrent

transactions on your Oracle server. These guidelines are appropriate for most application mixes.

To create rollback segments, use the CREATE ROLLBACK SEGMENT statement.

See Also: For information about the CREATE ROLLBACK SEGMENT statement, see the *Oracle8i SQL Reference.*

Choosing Sizes for Rollback Segments

The size of your rollback segment can also affect performance. Rollback segment size is determined by the storage parameters in the CREATE ROLLBACK SEGMENT statement. Your rollback segments must be large enough to hold the rollback entries for your transactions.

See Also: For information about choosing sizes for your rollback segments, see *Oracle8i Tuning.*

Choosing the Number of DB_BLOCK_LRU_LATCHES

Contention for the LRU (least recently used) latch can impede performance on symmetric multiprocessor (SMP) machines with a large number of CPUs. The LRU latch controls the replacement of buffers in the buffer cache. For SMP systems, Oracle automatically sets the number of LRU latches to be one half the number of CPUs on the system. For non-SMP systems, one LRU latch is sufficient.

You can specify the number of LRU latches on your system with the initialization parameter DB_BLOCK_LRU_LATCHES. This parameter sets the maximum value for the desired number of LRU latches. Each LRU latch will control a set of buffers and Oracle balances allocation of replacement buffers among the sets.

See Also: For more information on LRU latches, see *Oracle8i Tuning.*

Distributing I/O

Proper distribution of I/O can improve database performance dramatically. I/O can be distributed during installation of Oracle. Distributing I/O during installation can reduce the need to distribute I/O later when Oracle is running.

There are several ways to distribute I/O when you install Oracle:

- redo log file placement
- datafile placement
- separation of tables and indexes

- density of data (rows per data block)

See Also: For information about ways to distribute I/O, see *Oracle8i Tuning*.

3

Starting Up and Shutting Down

This chapter describes the procedures for starting and stopping an Oracle database, and includes the following topics:

- Starting Up a Database
- Altering Database Availability
- Shutting Down a Database
- Suspending and Resuming a Database
- Using Parameter Files

Starting Up a Database

This section includes the following topics:

To start up a database or an instance from the command line, use SQL*Plus to connect to Oracle with administrator privileges and then issue the STARTUP command. You can also use Recovery Manager to execute STARTUP and SHUTDOWN commands. If you are using the Enterprise Manager GUI and prefer not to use the command line, refer to the *Oracle Enterprise Manager Administrator's Guide* for instructions.

You can start an instance and database in a variety of ways:

- start the instance without mounting a database
- start the instance and mount the database, but leave it closed
- start the instance, and mount and open the database in:
  - unrestricted mode (accessible to all users)
  - restricted mode (accessible to database administrators only)

Note: You cannot start a database instance if you are connected to the database via a multi-threaded server process.

In addition, you can force the instance to start, or start the instance and have complete media recovery begin immediately. If your operating system supports the Oracle Parallel Server, you may start an instance and mount the database in either exclusive or shared mode.

See Also: For more information about starting a database in an OPS environment, see *Oracle8i Parallel Server Concepts and Administration.*

For more information on SQL*Plus command syntax, see *SQL*Plus User's Guide and Reference.*

For more information about Recovery Manager commands, see the *Oracle8i Backup and Recovery Guide.*

Preparing to Start an Instance

You need to perform several tasks before attempting to start an instance.

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

   ```
   sqlplus /nolog
   ```

2. Connect to Oracle as SYSDBA:

   ```
   connect username/password as sysdba
   ```

   Note that you cannot be connected via a multi-threaded server.

3. When you enter a STARTUP command, specify the database name and full path of the parameter file:

   ```
   STARTUP database_name PFILE=myinit.ora
   ```

   If you do not specify the PFILE option, Oracle uses the standard parameter file. If you do not specify a database name Oracle uses the value for DB_NAME in the parameter file that starts the instance.

See Also: The use of filenames is specific to your operating system. See your operating system-specific Oracle documentation.

For information about the DB_NAME parameter, see *Oracle8i Reference*.

Starting an Instance: Scenarios

The following scenarios describe the many ways in which you can start up an instance.

> **Note:** You may 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 parameter does not exist or cannot be opened when you attempt to mount a database, Oracle 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, Oracle returns a warning message and does not open the database.

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 option:

```
STARTUP NOMOUNT;
```

Starting an Instance and Mounting a Database

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

- renaming datafiles
- adding, dropping, or renaming redo log files
- enabling and disabling redo log archiving options
- performing full database recovery

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

```
STARTUP MOUNT;
```

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 and then mount and open the database by using the STARTUP command by itself:

```
STARTUP;
```

Restricting Access to a Database at Startup

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

- perform structure maintenance, such as rebuilding indexes
- perform an export or import of database data
- perform a data load (with SQL*Loader)
- temporarily prevent typical users from using data

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.

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

```
STARTUP RESTRICT;
```

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

See Also: For more information on the ALTER SYSTEM statement, see the *Oracle8i SQL Reference*.

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 option:

```
STARTUP FORCE;
```

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

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 option:

```
STARTUP OPEN RECOVER;
```

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

Starting in Exclusive or Parallel Mode

If your Oracle server allows multiple instances to access a single database concurrently, choose whether to mount the database exclusively or in parallel. For example, open in parallel mode you can issue:

```
STARTUP OPEN sales PFILE=INITSALE.ORA PARALLEL;
```

Starting Up an Instance and Database: Example

The following statement starts an instance using the parameter file INITSALE.ORA, mounts and opens the database named `sales` in exclusive mode, and restricts access to administrative personnel. The database administrator is already connected with administrator privileges.

```
STARTUP OPEN sales PFILE=INITSALE.ORA EXCLUSIVE RESTRICT;
```

Automatic Database Startup at Operating System Start

Many sites use procedures to enable automatic startup of one or more Oracle instances and databases immediately following a system start. The procedures for performing this task are specific to each operating system.

Starting Remote Instances

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

See Also: For more information about making a database available to non-privileged users, see "Restricting Access to an Open Database" on page 3-8.

For more information about recovering control files, database files, and redo logs, see Chapter 6, "Managing the Online Redo Log" and Chapter 7, "Managing Archived Redo Logs".

For more information about the side effects of aborting the current instance, see "Shutting Down with the ABORT Option" on page 3-12.

For more information about starting up in exclusive or parallel mode, see the *Oracle8i Parallel Server Concepts and Administration* manual.

For more information about the restrictions that apply when combining options of the STARTUP statement, see the *Oracle8i SQL Reference.*

For more information about automatic startup procedure topics, see your operating system-specific Oracle documentation.

Altering Database Availability

You can make a database partially available by starting an instance and opening a mounted database. The following sections explain how to alter a database's availability:

- 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.

When mounting the database, indicate whether to mount the database exclusively to this instance or concurrently to other instances.

To mount a database to a previously started instance, use the SQL statement ALTER DATABASE with the MOUNT option. Use the following statement when you want to mount a database in exclusive mode:

```
ALTER DATABASE MOUNT;
```

See Also: 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), see "Starting an Instance and Mounting a Database" on page 3-4.

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 SQL ALTER DATABASE statement with the OPEN option:

```
ALTER DATABASE OPEN;
```

After executing this statement, any valid Oracle 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 "state" modifications that don't generate redo. For example, you can take datafiles offline or bring them online since these operations do not effect data content.

Ideally, you open a database read-only when you alternate a standby database between read-only and recovery mode; note 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;
```

Note: You cannot use the RESETLOGS clause with a READ ONLY clause.

See Also: For more information about the ALTER DATABASE statement, see the *Oracle8i SQL Reference.*

For more conceptual details about opening a database in read-only mode, see *Oracle8i Concepts.*

Restricting Access to an Open Database

Under normal conditions, all users with the CREATE SESSION system privilege can connect to an instance. However, you can take an instance in and out of restricted mode. When an instance is in restricted mode, only users who have both the CREATE SESSION and RESTRICTED SESSION system privileges can connect to it. Typically, only administrators have the RESTRICTED SESSION system privilege.

Restricted mode is useful when you need to perform the following tasks:

- perform structure maintenance, such as rebuilding indexes
- perform an export or import of database data
- perform a data load (with SQL*Loader)

- temporarily prevent non-administrator users from using data

To place an instance in restricted mode, use the SQL statement ALTER SYSTEM with the ENABLE RESTRICTED SESSION option. After placing an instance in restricted mode, you might want to kill all current user sessions before performing any administrative tasks. To lift an instance from restricted mode, use ALTER SYSTEM with the DISABLE RESTRICTED SESSION option.

See Also: For more information about starting a database instance, and mounting and opening the database in restricted mode, see "Restricting Access to a Database at Startup" on page 3-4.

For more information on the ALTER SYSTEM statement, see the *Oracle8i SQL Reference*.

Shutting Down a Database

The following sections describe shutdown procedures:

- Shutting Down with the NORMAL Option
- Shutting Down with the IMMEDIATE Option
- Shutting Down with the TRANSACTIONAL Option
- Shutting Down with the ABORT Option

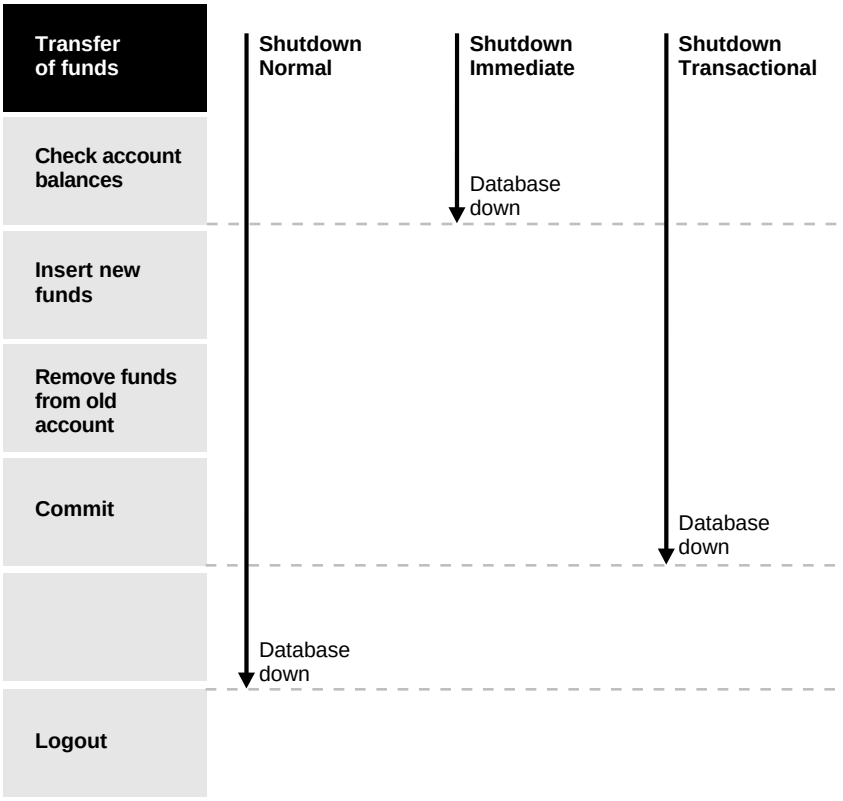
To initiate database shutdown, use the SQL*Plus SHUTDOWN command. Control is not returned to the session that initiates a database shutdown until shutdown is complete. Users who attempt connections while a shutdown is in progress receive a message like the following:

```
ORA-01090: shutdown in progress - connection is not permitted
```

Attention: You cannot shut down a database if you are connected to the database via a multi-threaded server process.

To shut down a database and instance, first connect as SYSOPER or SYSDBA. Figure 3–1 shows the sequence of events when the different SHUTDOWN commands are entered during a transfer of funds from one bank account to another.

Figure 3–1 Sequence of Events During Different Types of SHUTDOWN.



Shutting Down with the NORMAL Option

Normal database shutdown proceeds with the following conditions:

- No new connections are allowed after the statement is issued.
- Before the database is shut down, Oracle waits for all currently connected users to disconnect from the database.

- The next startup of the database will not require any instance recovery procedures.

To shut down a database in normal situations, use the SHUTDOWN command with the NORMAL option:

```
SHUTDOWN NORMAL;
```

Shutting Down with the IMMEDIATE Option

Use immediate database shutdown only in the following situations:

- A power shutdown is going to occur soon.
- The database or one of its applications is functioning irregularly.

Immediate database shutdown proceeds with the following conditions:

- Any uncommitted transactions are rolled back. (If long uncommitted transactions exist, this method of shutdown might not complete quickly, despite its name.)
- Oracle does not wait for users currently connected to the database to disconnect; Oracle implicitly rolls back active transactions and disconnects all connected users.

To shut down a database immediately, use the SHUTDOWN command with the IMMEDIATE option

```
SHUTDOWN IMMEDIATE;
```

Note: The SHUTDOWN IMMEDIATE statement disconnects all existing idle connections and shuts down the database. If, however, you have submitted processes (for example, inserts, selects or updates) that are awaiting results, the SHUTDOWN TRANSACTIONAL statement allows the process to complete before disconnecting.

Shutting Down with the TRANSACTIONAL Option

When you wish to perform a planned shutdown of an instance while allowing active transactions to complete first, use the SHUTDOWN command with the TRANSACTIONAL option:

```
SHUTDOWN TRANSACTIONAL;
```

After submitting this statement, no client can start a new transaction on this instance. If clients attempt to start a new transaction, they are disconnected. 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.

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 Option

You can shut down a database instantaneously by aborting the database's instance. If possible, perform this type of shutdown *only* in the following situations:

- The database or one of its applications is functioning irregularly *and* neither of the other types of shutdown work.
- 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.

Aborting an instance shuts down a database and yields the following results:

- Current client SQL statements being processed by Oracle are immediately terminated.
- Uncommitted transactions are not rolled back.
- Oracle does not wait for users currently connected to the database to disconnect; Oracle implicitly disconnects all connected users.

If *both* the normal and immediate shutdown options do not work, abort the current database instance immediately by issuing the SHUTDOWN command with the ABORT option:

```
SHUTDOWN ABORT;
```

Suspending and Resuming a Database

The ALTER SYSTEM SUSPEND statement suspends all I/O (datafile, control file, and file header) as well as queries, in all instances, enabling you to make copies of the database without having to handle ongoing transactions. Do not start a new instance while you suspend another instance, since the new instance will not be

suspended. Use the ALTER SYSTEM RESUME statement to resume normal database operations.

The suspend/resume feature is useful in systems that allow you to mirror a disk or file and then split the mirror. 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 handy substitute for normal shutdown operations, however, since copies of a suspended database can contain uncommitted updates.

Note that you can issue SUSPEND and RESUME statements from different instances. For example, if instances 1, 2, and 3 are running, and you issue a SUSPEND statement from instance 1, then you can issue the RESUME statement from instance 1, 2, or 3 with the same effect.

To Facilitate Mirror Splits Using SUSPEND and RESUME:

1. Place the database tablespaces in hot backup mode using the ALTER TABLESPACE BEGIN BACKUP statement.
2. If your mirror system has problems with splitting a mirror while disk writes are occurring, issue the ALTER SYSTEM SUSPEND statement.
3. Split your mirrors.
4. Issue the ALTER SYSTEM RESUME statement to resume your database.
5. Take the tablespaces out of hot backup mode using the ALTER TABLESPACE END BACKUP statement.
6. Copy the control file and archive the online redo logs as usual for a backup.

WARNING: Do not use the SUSPEND statement as a substitute for placing a tablespace in hot backup mode.

See Also: For more information about the ALTER SYSTEM SUSPEND/RESUME and ALTER TABLESPACE statements, see the *Oracle8i SQL Reference*.

Using Parameter Files

The following sections include information about how to use parameter files:

- The Sample Parameter File
- The Number of Parameter Files

- The Location of the Parameter File in Distributed Environments

To start an instance, Oracle must read a *parameter file*, which is a text file containing a list of instance configuration parameters. Often, although not always, this file is named INIT.ORA or INIT*sid*.ORA, where *sid* is operating system specific.

> **Note:** If you are using Oracle Enterprise Manager, see the *Oracle Enterprise Manager Administrator's Guide* information about using stored configurations as an alternative to the initialization parameter file.

You can edit parameter values in a parameter file with a basic text editor; however, editing methods are operating system-specific. For detailed information about initialization parameters, see the *Oracle8i Reference.*

Oracle treats string literals defined for National Language Support (NLS) parameters in the file as if they are in the database character set.

See Also: For more information about initialization parameter file, see your operating system-specific Oracle documentation.

The Sample Parameter File

A sample parameter file (INIT.ORA or INIT*sid*.ORA) is included in the Oracle distribution set. This sample file's parameters are adequate for initial installations of an Oracle database. After your system is operating and you have some experience with Oracle, you will probably want to change some parameter values.

See Also: For more information about optimizing a database's performance using the parameter file, see the *Oracle8i Tuning* manual.

The Number of Parameter Files

Each Oracle database has at least one parameter file that corresponds only to that database. This way, database-specific parameters such as DB_NAME and CONTROL_FILES in a given file always pertain to a particular database. It is also possible to have several different parameter files for a single database. For example, you can have several different parameter files for a single database so you can optimize the database's performance in different situations.

The Location of the Parameter File in Distributed Environments

The client you use to access the database must be able to read a database's parameter file to start a database's instance. Therefore, always store a database's parameter file on the computer executing the client.

In non-distributed processing installations, the same computer executes Oracle and the client. This computer already has the parameter file stored on one of its disk drives. In distributed processing installations, however, local client workstations can administer a database stored on a remote machine. In this type of configuration, the local client machines must each store a copy of the parameter file for the corresponding databases.

See Also: For more information about using administering Oracle in a distributed environment, see *Oracle8i Distributed Database Systems*.

Part II

Oracle Server Configuration

4

Managing Oracle Processes

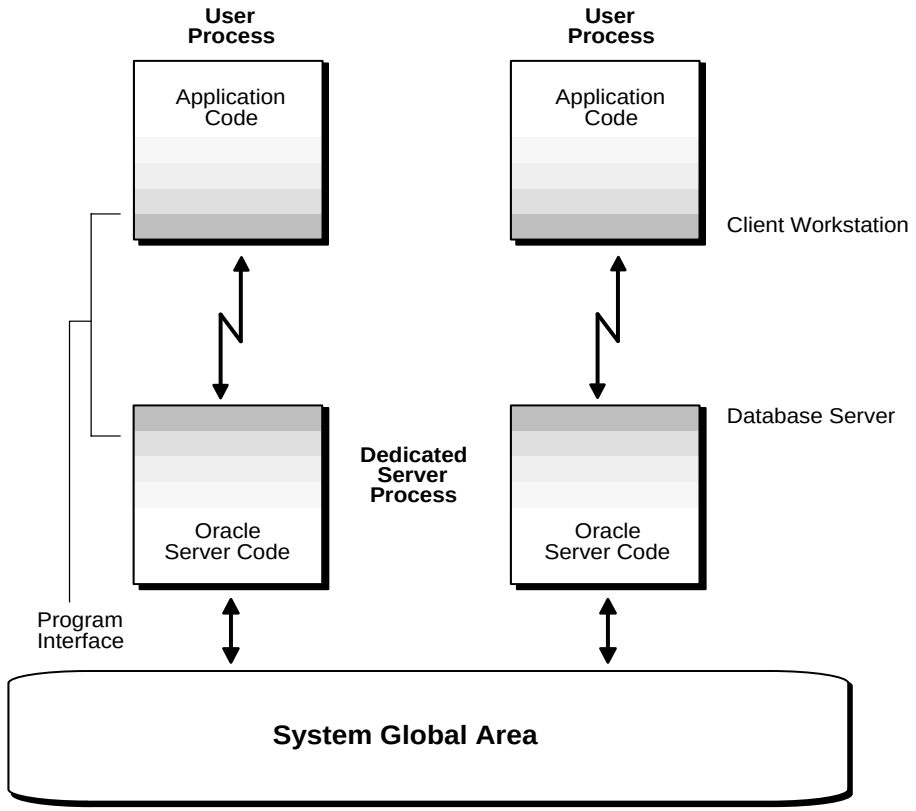
This chapter describes how to manage the processes of an Oracle instance, and includes the following topics:

- Setting Up Server Processes
- Configuring Oracle for Multi-Threaded Server Architecture
- Modifying Server Processes
- Tracking Oracle Processes
- Managing Processes for the Parallel Query Option
- Managing Processes for External Procedures
- Terminating Sessions

Setting Up Server Processes

When a user process executes the database application, and a separate, distinct server process executes the associated Oracle server on behalf of each user, the separate server process is a *dedicated server process* (see Figure 4–1). Oracle is automatically installed for this configuration. If your operating system can support Oracle in this configuration, it may also support multi-threaded server processes.

Figure 4–1 Oracle Dedicated Server Processes



When to Connect to a Dedicated Server Process

If possible, users should connect to an instance via a dispatcher. This 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 Enterprise Manager to start up, shut down, or perform media recovery on a database
- to use Recovery Manager to back up, restore or recover a database

To request a dedicated server connection, users must include the SERVER=DEDICATED clause in their Net8 TNS connect string.

See Also: For a complete description of Net8 connect string syntax, see your operating system-specific Oracle documentation and your *Net8 Administrator's Guide*.

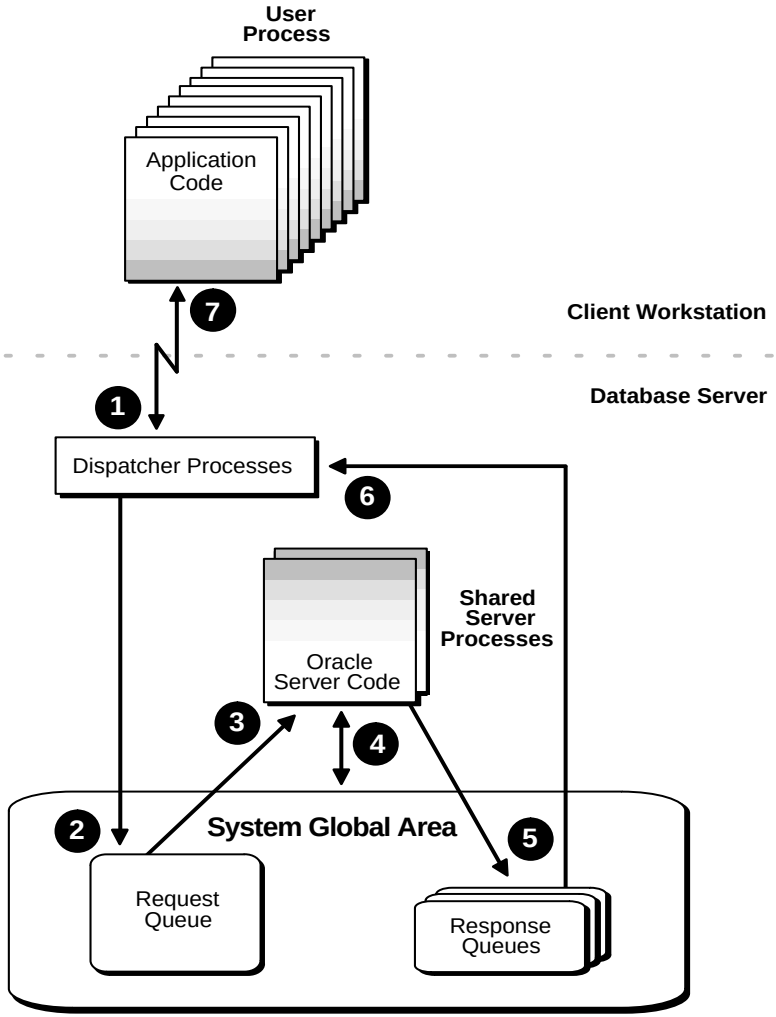
For more information about initialization parameters and parameter files, see the *Oracle8i Reference*.

Configuring Oracle for Multi-Threaded Server Architecture

Consider an order entry system with dedicated server processes. A customer places an order as a clerk enters the order into the database. For most of the transaction, the clerk is on the telephone talking to the customer and the server process dedicated to the clerk's user process remains idle. The server process is not needed during most of the transaction, and the system is slower for other clerks entering orders because the idle server process is holding system resources.

The *multi-threaded server* architecture eliminates the need for a dedicated server process for each connection (see Figure 4–2). A small number of shared server processes can perform the same amount of processing as many dedicated server processes. Also, since the amount of memory required for each user is relatively small, less memory and process management are required, and more users can be supported.

Figure 4–2 Oracle Multi-Threaded Sever Processes



To set up your system in a multi-threaded server configuration, start a network listener process and set the MTS_DISPATCHERS parameter (which is a required parameter that sets the initial number of dispatchers).

After setting this initialization parameter, restart the instance, which at this point will use the multi-threaded server configuration. The multi-threaded server architecture requires Net8. User processes targeting the multi-threaded server must connect through Net8, even if they are on the same machine as the Oracle instance.

See Also: For more information about starting and managing the network listener process, see *Oracle8i Distributed Database Systems* and the *Oracle Net8 Administrator's Guide.*

MTS_DISPATCHERS: Setting the Initial Number of Dispatchers (Required)

The number of dispatcher processes started at instance startup is controlled by the parameter MTS_DISPATCHERS. Estimate the number of dispatchers to start for each network protocol before instance startup.

When setting the MTS_DISPATCHERS parameter, you can include any valid protocol.

The appropriate number of dispatcher processes for each instance depends upon the performance you want from your database, the host operating system's limit on the number of connections per process, (which is operating system dependent) and the number of connections required per network protocol.

The instance must be able to provide as many connections as there are concurrent users on the database system. After instance startup, you can start more dispatcher processes if needed.

See Also: For more information about dispatcher processes, see "Adding and Removing Dispatcher Processes" on page 4-7.

Calculating the Initial Number of Dispatcher Processes

Once you know the number of possible connections per process for your operating system, calculate the initial number of dispatcher processes to create during instance startup, per network protocol, using the following formula.

$$\text{number of dispatchers} = \text{CEIL}\left(\frac{\text{maximum number of concurrent sessions}}{\text{connections per dispatcher}}\right)$$

Note: Here, *connections per dispatcher* is operating system dependent.

For example, assume that your system typically has 900 users concurrently connected via TCP/IP and 600 users connected via SPX, and supports 255 connections per process. In this case, the MTS_DISPATCHERS parameter should be set as follows:

```
MTS_DISPATCHERS = "(PROTOCOL=TCP) (DISPATCHERS=4)"
MTS_DISPATCHERS = "(PROTOCOL=SPX) (DISPATCHERS=3)"
```

Examples

Example 1 To force the IP address used for the dispatchers, enter the following:

```
MTS_DISPATCHERS="(ADDRESS=(PARTIAL=TRUE)(PROTOCOL=TCP)\
     (HOST=144.25.16.201))(DISPATCHERS=2)"
```

This will start two dispatchers that will listen in on HOST=144.25.16.201, which must be a card that is accessible to the dispatchers.

Example 2 To force the exact location of dispatchers, add the PORT as follows:

```
MTS_DISPATCHERS="(ADDRESS=(PARTIAL=TRUE)(PROTOCOL=TCP)\
     (HOST=144.25.16.201)(PORT=5000))(DISPATCHERS=1)"
MTS_DISPATCHERS="(ADDRESS=(PARTIAL=TRUE)(PROTOCOL=TCP)\
     (HOST=144.25.16.201)(PORT=5001))(DISPATCHERS=1)"
```

Note: You can specify multiple MTS_DISPATCHERS in the INIT.ORA file, but they must be adjacent to each other. Also, MTS_DISPATCHERS defaults to 1.

Modifying Server Processes

This section describes changes you can make after starting an instance, and includes the following topics:

- Changing the Minimum Number of Shared Server Processes
- Adding and Removing Dispatcher Processes

Changing the Minimum Number of Shared Server Processes

After starting an instance, you can change the minimum number of shared server processes by using the SQL command ALTER SYSTEM.

Oracle will eventually terminate servers that are idle when there are more shared servers than the minimum limit you specify.

If you set MTS_SERVERS to 0, Oracle will terminate all current servers when they become idle and will not start any new servers until you increase MTS_SERVERS. Thus, setting MTS_SERVERS to 0 may be used to effectively disables the multi-threaded server.

To control the minimum number of shared server processes, you must have the ALTER SYSTEM privilege.

The following statement sets the number of shared server processes to two:

```
ALTER SYSTEM SET MTS_SERVERS = 2
```

Adding and Removing Dispatcher Processes

You can control the number of dispatcher processes in the instance. If the V$QUEUE, V$DISPATCHER and V$DISPATCHER_RATE views indicate that the load on the dispatcher processes is consistently high, starting additional dispatcher processes to route user requests may improve performance; you can start additional dispatchers until the number of dispatchers equals MTS_MAX_DISPATCHER. In contrast, if the load on dispatchers is consistently low, reducing the number of dispatchers may improve performance.

To change the number of dispatcher processes, use the SQL command ALTER SYSTEM. Changing the number of dispatchers for a specific protocol has no effect on dispatchers for other protocols.

You can start new dispatcher processes for protocols specified in the MTS_DISPATCHERS parameter, or you may add new MTS_DISPATCHERS configurations. Therefore, you can add dispatchers for protocols for which there are dispatchers, and you can start dispatchers for protocols for which there are currently no dispatchers.

If you reduce the number of dispatchers for a particular protocol, the dispatchers are not immediately removed. Rather, Oracle eventually terminates dispatchers down to the limit you specify in MTS_DISPATCHERS.

To control the number of dispatcher processes, you must have the ALTER SYSTEM privilege.

The following example shows how to add a dispatcher process for the SPX protocol (where previously there was only one MTS_DISPATCHER configuration):

```
ALTER SYSTEM
   SET MTS_DISPATCHERS = '(INDEX=1) (PRO=SPX)';
```

See Also: For more information about tuning the multi-threaded server, see *Oracle8i Tuning*.

Tracking Oracle Processes

An Oracle instance can have many background processes, which you should track if possible. This section describes how to track these processes, and includes the following topics:

- Monitoring the Processes of an Oracle Instance
- Trace Files, the ALERT File, and Background Processes
- Starting the Checkpoint Process

See Also: For more information about tuning Oracle processes, see *Oracle8i Tuning*.

Monitoring the Processes of an Oracle Instance

Monitors provide a means of tracking database activity and resource usage. You can operate several monitors simultaneously. Table 4–1 lists the Enterprise Manager monitors that can help you track Oracle processes:

Table 4–1 Enterprise Manager Monitors

Monitor Name	Description
Process	The Process monitor summarizes information about all Oracle processes, including client-server, user, server, and background processes, currently accessing the database via the current database instance.
Session	The Session monitor shows the session ID and status of each connected Oracle session.

Monitoring Locks

Table 4–2 describes two methods of monitoring locking information for ongoing transactions within an instance:

Table 4–2 Oracle Monitoring Facilities

Monitor Name	Description
Enterprise Manager Monitors	The Monitor feature of Enterprise Manager / GUI provides two monitors for displaying lock information for an instance: Lock and Latch Monitors.
UTLLOCKT.SQL	The UTLLOCKT.SQL script displays a simple character lock wait-for graph in tree-structured fashion. Using an ad hoc query tool (such as Enterprise Manager or SQL*Plus), the script prints the sessions in the system that are waiting for locks and the corresponding blocking locks. The location of this script file is operating system dependent; see your operating system-specific Oracle documentation. (A second script, CATBLOCK.SQL, creates the lock views that UTLLOCKT.SQL needs, so you must run it before running UTLLOCKT.SQL.)

Monitoring Dynamic Performance Tables

The following views, created on the dynamic performance tables, are useful for monitoring Oracle instance processes:

View (Monitor) Name	Description
V$CIRCUIT	Contains information about virtual circuits, which are user connections through dispatchers and servers.
V$QUEUE	Contains information about the multi-threaded message queues.
V$DISPATCHER	Contains information about dispatcher processes.
V$DISPATCHER_RATE	Contains rate statistics for dispatcher processes.
V$SHARED_SERVER	Contains information about shared server processes.
V$SQLAREA	Contains statistics about shared SQL area and contains one row per SQL string. Also provides statistics about SQL statements that are in memory, parsed, and ready for execution.
V$SESS_IO	Contains I/O statistics for each user session.
V$LATCH	Contains statistics for non-parent latches and summary statistics for parent latches.
V$SYSSTAT	Contains system statistics.

Following is a typical query of one of the dynamic performance tables, V$DISPATCHER. The output displays the processing load on each dispatcher process in the system:

```
SELECT (busy/(busy + idle)) * 100 "% OF TIME BUSY"
   FROM v$dispatcher;
```

Distinguishing Oracle Background Processes from Operating System Background Processes

When you run many Oracle databases concurrently on one computer, Oracle provides a mechanism for naming the processes of an instance. The background process names are prefixed by an instance identifier to distinguish the set of processes for each instance.

For example, an instance named TEST might have background processes with the following names:

- ORA_TEST_DBWR
- ORA_TEST_LGWR
- ORA_TEST_SMON
- ORA_TEST_PMON
- ORA_TEST_RECO
- ORA_TEST_LCK0
- ORA_TEST_ARCH
- ORA_TEST_D000
- ORA_TEST_S000
- ORA_TEST_S001

See Also: For more information about views and dynamic performance tables see the *Oracle8i Reference*.

For more information about the instance identifier and the format of the Oracle process names, see your operating system-specific Oracle documentation.

Trace Files, the ALERT File, and Background Processes

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, while other information is for Oracle WorldWide Support. Trace file information is also used to tune applications and instances.

The *ALERT* file is a special trace file. The ALERT file of a database is a chronological log of messages and errors, which includes the following:

- all internal errors (ORA-600), block corruption errors (ORA-1578), and deadlock errors (ORA-60) that occur
- administrative operations, such as CREATE / ALTER / DROP DATABASE / TABLESPACE / ROLLBACK SEGMENT SQL statements and STARTUP, SHUTDOWN, and ARCHIVE LOG
- several messages and errors relating to the functions of shared server and dispatcher processes
- errors occurring during the automatic refresh of a snapshot

- the values of all initialization parameters at the time the database and instance start

Oracle uses the ALERT file to keep a log of these special operations as an alternative to displaying such information on an operator's console (although many systems display information on the console). If an operation is successful, a "completed" message is written in the ALERT file, along with a timestamp.

Using the Trace Files

You can periodically check the ALERT file and other trace files of an instance to see if the background processes have encountered errors. For example, when the Log Writer process (LGWR) cannot write to a member of a group, an error message indicating the nature of the problem is written to the LGWR trace file and the database's ALERT file. If you see such error messages, a media or I/O problem has occurred, and should be corrected immediately.

Oracle also writes values of initialization parameters to the ALERT file, in addition to other important statistics. For example, when you shut down an instance normally or immediately (but do not abort), Oracle writes the highest number of sessions concurrently connected to the instance, since the instance started, to the ALERT file. You can use this number to see if you need to upgrade your Oracle session license.

Specifying the Location of Trace Files

All trace files for background processes and the ALERT file are written to the destination specified by the initialization parameter BACKGROUND_DUMP_DEST. All trace files for server processes are written to the destination specified by the initialization parameter USER_DUMP_DEST. The names of trace files are operating system specific, but usually include the name of the process writing the file (such as LGWR and RECO).

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. This limit is set as a 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 Oracle continues to append to the file. You can safely delete the ALERT file while the instance is running, although you might want to make an archived copy of it first.

Controlling When Oracle Writes to Trace Files

Background processes always write to a trace file when appropriate. However, trace files are written on behalf of server processes (in addition to being written to during internal errors) only if the initialization parameter SQL_TRACE is set to TRUE.

Regardless of the current value of SQL_TRACE, each session can enable or disable trace logging on behalf of the associated server process by using the SQL command ALTER SESSION with the SET SQL_TRACE parameter.

```
ALTER SESSION SET SQL_TRACE TRUE;
```

For the multi-threaded server, each session using a dispatcher is routed to a shared server process, and trace information is written to the server's 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's trace files. Because the SQL trace facility for server processes can cause significant system overhead, enable this feature only when collecting statistics.

See Also: For information about the names of trace files, see your operating system-specific Oracle documentation.

For complete information about the ALTER SESSION command, see the *Oracle8i SQL Reference.*

Starting the Checkpoint Process

If the Checkpoint process (CKPT) is not enabled, the Log Writer process (LGWR) is responsible for updating the headers of all control files and data files to reflect the latest checkpoint. To reduce the time necessary to complete a checkpoint, especially when a database is comprised of many data files, enable the CKPT background process by setting the CHECKPOINT_PROCESS parameter in the database's parameter file to TRUE. (The default is FALSE.)

Managing Processes for the Parallel Query Option

This section describes how, with the parallel query option, Oracle can perform parallel processing. In this configuration Oracle can divide the work of processing certain types of SQL statements among multiple query server processes. The following topics are included:

- Managing the Query Servers
- Variations in the Number of Query Server Processes

See Also: For more information about the parallel query option, see *Oracle8i Tuning*.

Managing the Query Servers

When you start your instance, the Oracle Server creates a pool of query server processes available for any query coordinator. Specify the number of query server processes that the Oracle Server creates at instance startup via the initialization parameter PARALLEL_MIN_SERVERS.

Query server processes remain associated with a statement throughout its execution phase. When the statement is completely processed, its query server processes become available to process other statements. The query coordinator process returns any resulting data to the user process issuing the statement.

Variations in the Number of Query Server Processes

If the volume of SQL statements processed concurrently by your instance changes drastically, the Oracle Server automatically changes the number of query server processes in the pool to accommodate this volume.

If this volume increases, the Oracle Server automatically creates additional query server processes to handle incoming statements. The maximum number of query server processes for your instance is specified by the initialization parameter PARALLEL_MAX_SERVERS.

If this volume subsequently decreases, the Oracle Server terminates a query server process if it has been idle for the period of time specified by the initialization parameter PARALLEL_SERVER_IDLE_TIME. The Oracle Server does not reduce the size of the pool below the value of PARALLEL_MIN_SERVERS, no matter how long the query server processes have been idle.

If all query servers in the pool are occupied and the maximum number of query servers has been started, a query coordinator processes the statement sequentially.

See Also: For more information about monitoring an instance's pool of query servers and determining the appropriate values of the initialization parameters, see *Oracle8i Tuning*.

Managing Processes for External Procedures

You may have shared libraries of C functions that you wish to call from an Oracle database. This section describes how to set up an environment for calling those external procedures.

> **Note:** Although not required, it is recommended that you perform these tasks during installation.

The database administrator grants execute privileges for appropriate libraries to application developers, who in turn create external procedures and grant execute privilege on the specific external procedures to other users.

To Set Up an Environment for Calling External Procedures

1. Edit the *tnsnames.ora* file by adding an entry that enables you to connect to the listener process (and subsequently, the EXTPROC process).
2. Edit the *listener.ora* file by adding an entry for the "external procedure listener."
3. Start a separate listener process to exclusively handle external procedures.
4. The EXTPROC process spawned by the listener inherits the operating system privileges of the listener, so Oracle strongly recommends that you restrict the privileges for the separate listener process. The process should not have permission to read or write to database files, or the Oracle server address space.

   Also, the owner of this separate listener process should not be "oracle" (which is the default owner of the server executable and database files).
5. If not already installed, place the **extproc** executable in $ORACLE_HOME/bin.

Be aware that 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). These symbols are not resolved and can cause your external procedure to fail.

Sample Entry in *tnsnames.ora*

The following is a sample entry for the external procedure listener in *tnsnames.ora*.

```
extproc_connection_data = (DESCRIPTION =
                            (ADDRESS = (PROTOCOL=IPC)
                                       (KEY=extproc_key)
                             )
                           (CONNECT_DATA = (SID = extproc_agent)
```

```
                                    )
```

In this example, and all callouts for external procedures, the entry name **extproc_connection_data** cannot be changed; it must be entered exactly as it appears here. The key you specify—in this case **extproc_key**—must match the KEY you specify in the *listener.ora* file. Additionally, the SID name you specify—in this case **extproc_agent**—must match the SID_NAME entry in the *listener.ora* file.

Sample Entry in *listener.ora*

The following is a sample entry for the external procedure in *listener.ora*.

```
EXTERNAL_PROCEDURE_LISTENER =

(ADDRESS_LIST =
    (ADDRESS = (PROTOCOL=ipc)
               (KEY=extproc_key)
     )
)
...
SID_LIST_EXTERNAL_PROCEDURE_LISTENER =

(SID_LIST =
     (SID_DESC = (SID_NAME=extproc_agent)
                 (ORACLE_HOME=/oracle)
                 (PROGRAM=extproc)
      )
)
```

In this example, the PROGRAM must be **extproc**, and cannot be changed; it must be entered exactly as it appears in this example. The SID_NAME must match the SID name in the *tnsnames.ora* file. The ORACLE_HOME must be set to the directory where your Oracle software is installed. The **extproc** executable must reside in $ORACLE_HOME/bin.

See Also: For more information about external procedures, see the *PL/SQL User's Guide and Reference.*

Terminating Sessions

In some situations, you might want to terminate current user sessions. For example, you might want to perform an administrative operation and need to terminate all non-administrative sessions.

This section describes the various aspects of terminating sessions, and includes the following topics:

When a session is terminated, the session's transaction is rolled back and resources (such as locks and memory areas) held by the session are immediately released and available to other sessions.

Terminate a current session using the SQL statement ALTER SYSTEM KILL SESSION.

The following statement terminates the session whose SID 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's index number and serial number. To identify the index (SID) and serial numbers of a session, query the V$SESSION dynamic performance table.

The following query identifies all sessions for the user JWARD:

```
SELECT sid, serial#
  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. A session is INACTIVE if it is not making a SQL call to Oracle.

See Also: For a complete description of the status values for a session, see *Oracle8i Tuning*.

Terminating an Active Session

If a user session is making a SQL call to Oracle (is ACTIVE) when it is terminated, 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 returns the following message:

```
ORA-01012: not logged on
```

If an active session cannot be interrupted (for example, it is performing network I/O or rolling back a transaction), the 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 (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, STATUS in the view V$SESSION is "KILLED." The row for the terminated session is removed from V$SESSION after the user attempts to use the session again and receives the ORA-00028 message.

In the following example, the administrator terminates an inactive session:

```
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.
```

5

Managing Control Files

This chapter explains how to create and maintain the control files for your database, and includes the following topics:

- Guidelines for Control Files
- Creating Control Files
- Troubleshooting After Creating Control Files
- Dropping Control Files

Guidelines for Control Files

This section describes guidelines you can use to manage the control files for a database, and includes the following topics:

Name Control Files

Assign control file names via the CONTROL_FILES initialization parameter in the database's parameter file. CONTROL_FILES indicates one or more names of control files separated by commas. The instance startup procedure recognizes and opens all the listed files. The instance maintains all listed control files during database operation.

During database operation, Oracle writes to all necessary files listed for the CONTROL_FILES parameter.

Multiplex Control Files on Different Disks

Every Oracle database should have at least two control files, each stored on a different 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 an intact copy of the control file and the instance can be restarted; no media recovery is required.

Behavior of Multiplexed Control Files

The following list describes the behavior of multiplexed control files:

- Two or more filenames are listed for the initialization parameter CONTROL_FILES in the database's parameter file.
- The first file listed in the CONTROL_FILES parameter is the only file read by the Oracle Server during database operation.
- If any of the control files become unavailable during database operation, the instance becomes inoperable and should be aborted.

The only disadvantage of having multiple control files is that all operations that update the control files (such as adding a datafile or checkpointing the database) can take slightly longer. However, this difference is usually insignificant (especially for operating systems that can perform multiple, concurrent writes) and does not justify using only a single control file.

Attention: Oracle strongly recommends that your database has a minimum of two control files on different disks.

Place Control Files Appropriately

Each copy of a control file should be stored on a different disk drive. Furthermore, a control file copy should be stored on every disk drive that stores members of online redo log groups, if the online redo log is multiplexed. By storing control files in these locations, you minimize the risk that all control files and all groups of the online redo log will be lost in a single disk failure.

Manage the Size of Control Files

The main determinants of a control file's size 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: The maximum control file size is operating system specific. See your operating system-specific Oracle documentation for more information.

Creating Control Files

Every Oracle database has a *control file*. A control files records the physical structure of the database and contains:

- the database name
- names and locations of associated databases and online redo log files
- the timestamp of the database creation
- the current log sequence number
- checkpoint information

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 must be created during database creation. On some operating systems, Oracle creates multiple copies. You should create two or more copies of the control file during database creation. You might also need to create control files later, if you lose control files or want to change particular settings in the control files.

This section describes ways to create control files, and includes the following topics:

- Creating Initial Control Files
- Creating Additional Control File Copies, and Renaming and Relocating Control Files
- New Control Files
- Creating New Control Files

Creating Initial Control Files

You create the initial control files of an Oracle database by specifying one or more control filenames in the CONTROL_FILES parameter in the parameter file used during database creation. The filenames specified in CONTROL_FILES should be fully specified. Filename specification is operating system-specific.

If files with the specified names currently exist at the time of database creation, you must specify the CONTROLFILE REUSE parameter in the CREATE DATABASE statement, or else an error occurs. Also, if the size of the old control file differs from that of the new one, you cannot use the REUSE option. The size of the control file changes between some releases of Oracle, 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.

If you do not specify files for CONTROL_FILES before database creation, Oracle uses a default filename. The default name is also operating system-specific.

You can subsequently change the value of the CONTROL_FILES parameter to add more control files or to change the names or locations of existing control files.

See Also: For more information about specifying control files, see your operating system-specific Oracle documentation.

Creating Additional Control File Copies, and Renaming and Relocating Control Files

You add a new control file by copying an existing file to a new location and adding the file's 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's name in the control file list.

In both cases, to guarantee that control files do not change during the procedure, shut down the instance before copying the control file.

To Multiplex or Move Additional Copies of the Current Control Files

1. Shut down the database.
2. Copy an existing control file to a different location, using operating system commands.
3. Edit the CONTROL_FILES parameter in the database's parameter file to add the new control file's name, or to change the existing control filename.
4. Restart the database.

New Control Files

You can create a new control file for a database using the CREATE CONTROLFILE statement. This is recommended 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 settings originally specified in the CREATE DATABASE statement, including the database's name, MAXLOGFILES, MAXLOGMEMBERS, MAXLOGHISTORY, MAXDATAFILES, and MAXINSTANCES.

  For example, you might need to change a database's name if it conflicts with another database's name in a distributed environment. As another example, you might need to change one of the previously mentioned parameters if the original setting is too low.

The following statement creates a new control file for the PROD database (formerly a database that used a different database name):

```
CREATE CONTROLFILE
   SET DATABASE prod
   LOGFILE GROUP 1 ('logfile1A', 'logfile1B') SIZE 50K,
    GROUP 2 ('logfile2A', 'logfile2B') SIZE 50K
```

```
  NORESETLOGS
  DATAFILE 'datafile1' SIZE 3M, 'datafile2' SIZE 5M
MAXLOGFILES 50
MAXLOGMEMBERS 3
MAXDATAFILES 200
MAXINSTANCES 6
ARCHIVELOG;
```

WARNING: The CREATE CONTROLFILE statement can potentially damage specified datafiles and online redo log files; omitting a filename can cause loss of the data in that file, or loss of access to the entire database. Employ caution when using this command and be sure to follow the steps in the next section.

See Also: For more information about the CREATE CONTROLFILE statement, see the *Oracle8i SQL Reference.*

Creating New Control Files

This section provides step-by-step instructions for creating new control files.

To Create New Control Files

1. Make a list of all datafiles and online redo log files of the database.

   If you followed the recommendations for database backups, you should already have a list of datafiles and online redo log files that reflect the current structure of the database.

   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 online 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 with normal priority, if possible. Use the IMMEDIATE or ABORT options only as a last resort.

3. Back up all datafiles and online redo log files of the database.

4. Start up an new instance, but do not mount or open the database.

5. Create a new control file for the database using the CREATE CONTROLFILE statement.

   When creating the new control file, select the RESETLOGS option if you have lost any online redo log groups in addition to the control files. In this case, you will need to recover from the loss of the redo logs (Step 8). You must also specify the RESETLOGS option if you have renamed the database. Otherwise, select the NORESETLOGS option.

6. Store a backup of the new control file on an offline storage device.

7. Edit the parameter files of the database.

   Edit the parameter files of the database to indicate all of the control files created in Step 5 and Step 6 (not including the backup control file) in the CONTROL_FILES parameter.

8. Recover the database if necessary.

   If you are creating the control file as part of recovery, recover the database. If the new control file was created using the NORESETLOGS option (Step 5), you can recover the database with complete, closed database recovery.

   If the new control file was created using the RESETLOGS option, 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.

9. Open the database.

   Open the database using one of the following methods:

- If you did not perform recovery, open the database normally.
- If you performed complete, closed database recovery in Step 8, start up the database.
- If you specified RESETLOGS when creating the control file, use the ALTER DATABASE statement, indicating RESETLOGS.

The database is now open and available for use.

See Also: See the *Oracle8i Backup and Recovery Guide* for more information about:

- listing database files
- backing up all datafiles and online redo log files of the database
- recovering online or archived redo log files
- closed database recovery

Troubleshooting After Creating Control Files

After issuing the CREATE CONTROLFILE statement, you may encounter some common errors. This section describes the most common control file usage errors, and includes the following topics:

- 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 log to see if Oracle has detected inconsistencies between the data dictionary and the control file, such as a datafile that the data dictionary includes but the control file does not list.

If a datafile exists in the data dictionary but not in the new control file, Oracle 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.

In the following two cases only, the actual datafile corresponding to MISSING*nnnn* can be made accessible by renaming MISSING*nnnn* to point to it.

Case 1: The new control file was created using the CREATE CONTROLFILE statement with the NORESETLOGS option, thus allowing the database to be opened without using the RESETLOGS option. This would be possible only if all online redo logs are available.

Case 2: It was necessary to use the RESETLOGS option of the CREATE CONTROLFILE statement, thus forcing the database to be opened using the RESETLOGS option, but the actual datafile corresponding to MISSING*nnnn* was read-only or offline normal.

If, on the other hand, it was necessary to open the database using the RESETLOGS option, and MISSING*nnnn* corresponds to a datafile that was not read-only or offline normal, then the rename operation cannot be used to make the datafile accessible (since the datafile requires media recovery that is precluded by the results of RESETLOGS). In this case, the tablespace containing the datafile must be dropped.

In contrast, if a datafile indicated in the control file is not present in the data dictionary, Oracle removes references to it from the new control file. In both cases,

Oracle includes an explanatory message in the ALERT file to let you know what it found.

Handling Errors During CREATE CONTROLFILE

If Oracle 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 and repeat the procedure from Step 4, using the correct filenames.

Dropping Control Files

You can drop control files from the database. For example, you might want to do so if the location of a control file is inappropriate. Remember that the database must have at least two control files at all times.

1. Shut down the database.
2. Edit the CONTROL_FILES parameter in the database's parameter file to delete the old control file's name.
3. Restart the database.

> **WARNING: 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.**

6

Managing the Online Redo Log

This chapter explains how to manage the online redo log and includes the following topics:

- What Is the Online Redo Log?
- Planning the Online Redo Log
- Creating Online Redo Log Groups and Members
- Renaming and Relocating Online Redo Log Members
- Dropping Online Redo Log Groups and Members
- Forcing Log Switches
- Verifying Blocks in Redo Log Files
- Clearing an Online Redo Log File
- Listing Information about the Online Redo Log

See Also: For more information about managing the online redo logs of the instances when using Oracle Parallel Server, see *Oracle8i Parallel Server Concepts and Administration*.

To learn how checkpoints and the redo log impact instance recovery, see *Oracle8i Tuning*.

What Is the Online Redo Log?

The most crucial structure for recovery operations is the online redo log, which consists of two or more pre-allocated files that store all changes made to the database as they occur. Every instance of an Oracle database has an associated online redo log to protect the database in case of an instance failure.

Note: Oracle does not recommend backing up the online redo log.

Redo Threads

Each database instance has its own online *redo log groups*. These online redo log groups, multiplexed or not, are called an instance's *thread* of online redo. In typical configurations, only one database instance accesses an Oracle database, so only one thread is present. When running the Oracle Parallel Server, however, two or more instances concurrently access a single database; each instance has its own thread.

This chapter describes how to configure and manage the online redo log when the Oracle Parallel Server is *not* used. Hence, the thread number can be assumed to be 1 in all discussions and examples of commands.

See Also: For complete information about configuring the online redo log with the Oracle Parallel Server, see *Oracle8i Parallel Server Concepts and Administration*.

Online Redo Log Contents

Online 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 rollback segment data block, and the transaction table of the rollback segments.

Redo entries record data that you can use to reconstruct all changes made to the database, including the rollback segments. Therefore, the online redo log also protects rollback data. When you recover the database using redo data, Oracle 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 and are written to one of the online redo log files by the Oracle background process Log Writer (LGWR). Whenever a transaction is committed, LGWR writes the transaction's redo records from the redo log buffer of the SGA to an online redo log

file, and a *system change number* (SCN) is assigned to identify the redo records for each committed transaction. Only once 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 an online 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 an online redo log file, even though some redo records may not be committed. If necessary, Oracle can roll back these changes.

How Oracle Writes to the Online Redo Log

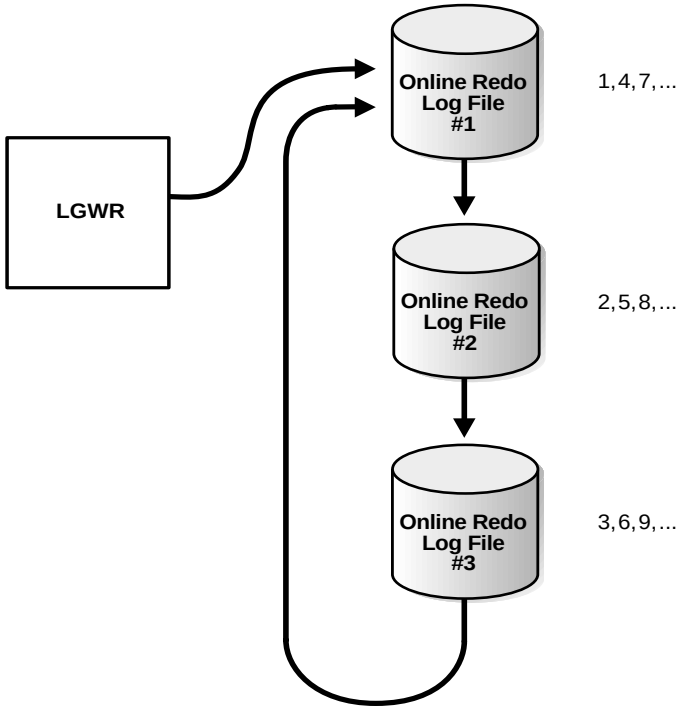
The online redo log of a database consists of two or more online redo log files. Oracle requires a minimum of two files to guarantee that one is always available for writing while the other is being archived (if in ARCHIVELOG mode).

LGWR writes to online redo log files in a circular fashion; when the current online redo log file fills, LGWR begins writing to the next available online redo log file. When the last available online redo log file is filled, LGWR returns to the first online redo log file and writes to it, starting the cycle again. Figure 6–1 illustrates the circular writing of the online redo log file. The numbers next to each line indicate the sequence in which LGWR writes to each online redo log file.

Filled online redo log files are available to LGWR for re-use depending on whether archiving is enabled:

- If archiving is disabled (NOARCHIVELOG mode), a filled online redo log file is available once the changes recorded in it have been written to the datafiles.
- If archiving is enabled (ARCHIVELOG mode), a filled online redo log file is available to LGWR once the changes recorded in it have been written to the datafiles *and* once the file has been archived.

Figure 6–1 Circular Use of Online Redo Log Files by LGWR



Active (Current) and Inactive Online Redo Log Files

At any given time, Oracle uses only one of the online redo log files to store redo records written from the redo log buffer. The online redo log file that LGWR is actively writing is called the *current* online redo log file.

Online redo log files that are required for instance recovery are called *active* online redo log files. Online redo log files that are not required for instance recovery are called *inactive*.

If you have enabled archiving, Oracle cannot re-use or overwrite an active online log file until ARC*n* has archived its contents. If archiving is disabled, when the last online redo log file fills, writing continues by overwriting the first available active file.

Log Switches and Log Sequence Numbers

A *log switch* is the point at which Oracle ends writing to one online redo log file and begins writing to another. A log switch always occurs when the current online redo log file is completely filled and writing must continue to the next online redo log file. You can also force log switches manually.

Oracle assigns each online redo log file a new *log sequence number* every time that a log switch occurs and LGWR begins writing to it. If Oracle archives online redo log files, the archived log retains its log sequence number. The online 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, Oracle properly applies redo log files in ascending order by using the log sequence number of necessary archived and online redo log files.

Planning the Online Redo Log

This section describes guidelines you should consider when configuring a database instance's online redo log, and includes the following topics:

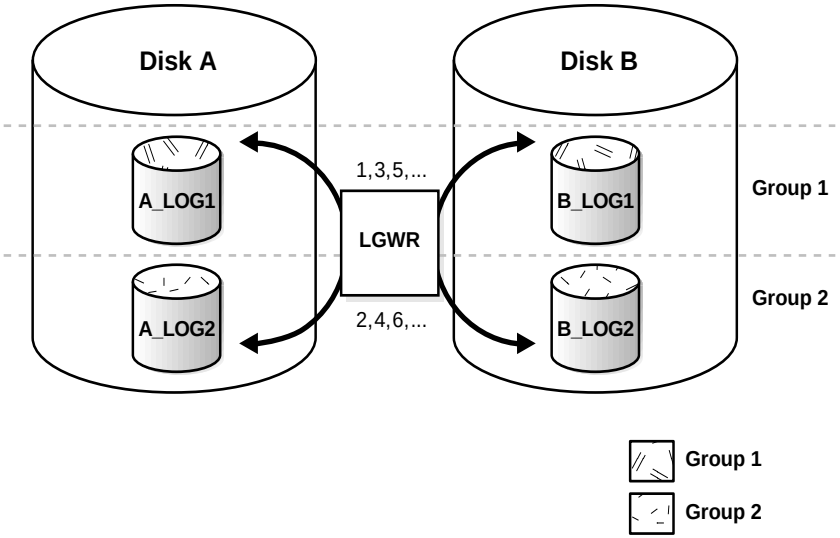
- Multiplexing Online Redo Log Files
- Placing Online Redo Log Members on Different Disks
- Setting the Size of Online Redo Log Members
- Choosing the Number of Online Redo Log Files

Multiplexing Online Redo Log Files

Oracle provides the capability to *multiplex* an instance's online redo log files to safeguard against damage to its online redo log files. When multiplexing online redo log files, LGWR concurrently writes the same redo log information to multiple identical online 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 Online Redo Log Files



The corresponding online redo log files are called *groups*. Each online 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.

Notice that each member of a group is concurrently active, or, concurrently written to by LGWR, as indicated by the identical log sequence numbers assigned by LGWR. In Figure 6–2, first LGWR writes to A_LOG1 in conjunction with B_LOG1, then A_LOG2 in conjunction with B_LOG2, etc. LGWR never writes concurrently to members of different groups (for example, to A_LOG1 and B_LOG2).

Responding to Online Redo Log Failure

Whenever LGWR cannot write to a member of a group, Oracle marks that member as stale and writes an error message to the LGWR trace file and to the database's alert file to indicate the problem with the inaccessible files. LGWR reacts differently when certain online redo log members are unavailable, depending on the reason for the unavailability.

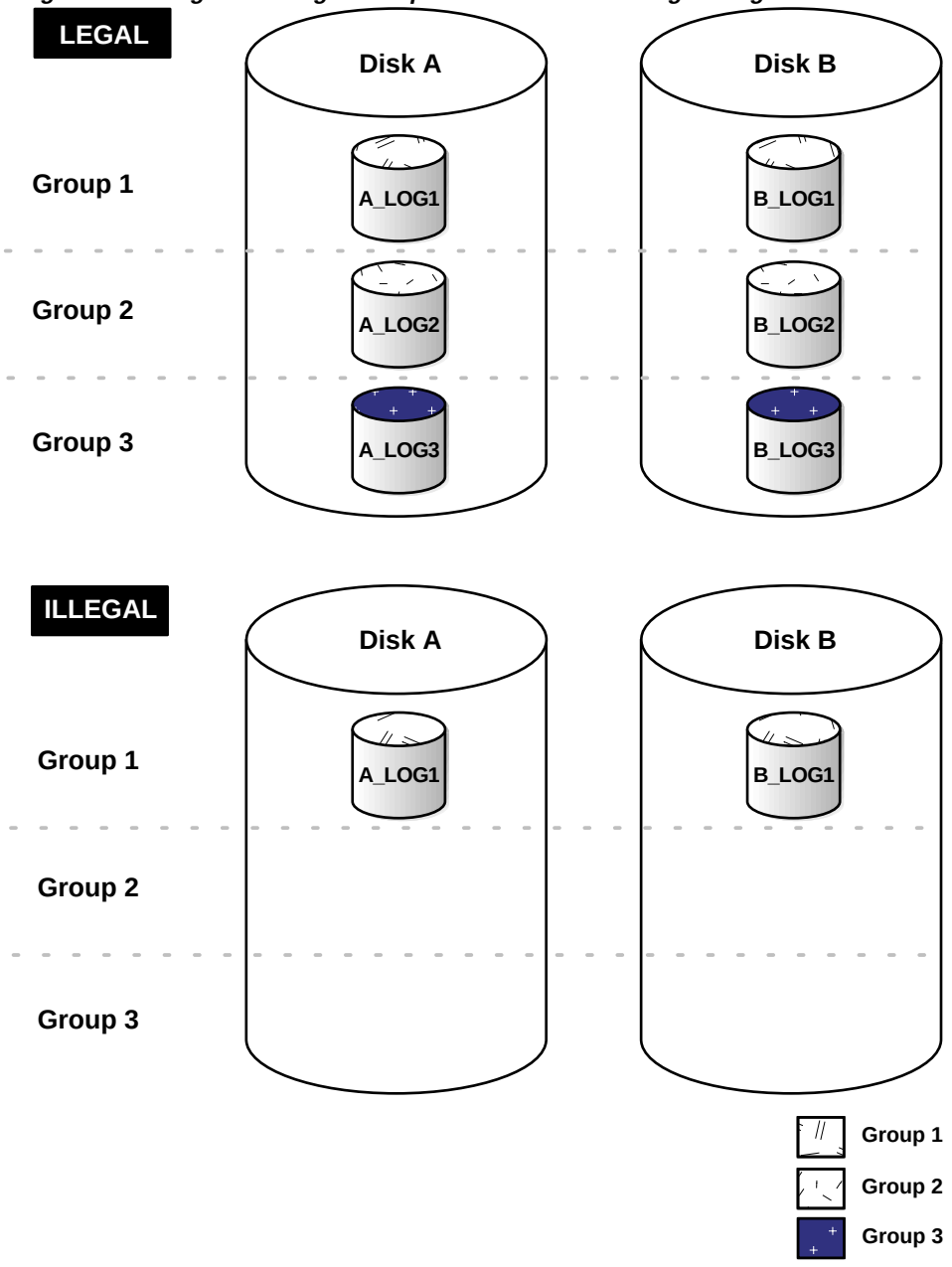
If	Then
LGWR can successfully write to at least one member in a group	Writing proceeds as normal; LGWR simply writes to the available members of a group and ignores the unavailable members.
LGWR cannot access the next group at a log switch because the group needs to be archived	Database operation temporarily 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 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 an online redo log file. If the database checkpoint has moved beyond the lost redo log (which is not the current log in this example), media recovery is not necessary since Oracle has saved the data recorded in the redo log to the datafiles. Simply drop the inaccessible redo log group. If Oracle did not archive the bad log, use ALTER DATABASE CLEAR UNARCHIVED LOG to disable archiving before the log can be dropped.
If all members of a group suddenly become inaccessible to LGWR while it is writing to them	Oracle 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 only need to turn the drive back on and let Oracle perform instance recovery.

Legal and Illegal Configurations

To safeguard against a single point of online redo log failure, a multiplexed online redo log is ideally symmetrical: all groups of the online redo log have the same number of members. Nevertheless, Oracle does not *require* that a multiplexed online redo log be symmetrical. For example, one group can have only one member, while other groups have two members. This configuration protects against disk failures that temporarily affect some online redo log members but leave others intact.

The only requirement for an instance's online redo log is that it have at least two groups. Figure 6–3 shows legal and illegal multiplexed online redo log configurations. The second configuration is illegal because it has only one group.

Figure 6–3 Legal and Illegal Multiplexed Online Redo Log Configuration



Placing Online Redo Log Members on Different Disks

When setting up a multiplexed online redo log, place members of a group on different 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 online 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 online redo log members, place each member on a different disk and set your archiving destination to a fifth disk. Consequently, there is never contention between LGWR (writing to the members) and ARC*n* (reading the members).

Datafiles and online redo log files should also be on different disks to reduce contention in writing data blocks and redo records.

See Also: For more information about how the online redo log affects backup and recovery, see *Oracle8i Backup and Recovery Guide*.

Setting the Size of Online Redo Log Members

When setting the size of online redo log files, consider whether you will be archiving the redo log. Online 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 online redo log group can fit on a tape and 49% of the tape's storage capacity remains unused. In this case, it is better to decrease the size of the online redo log files slightly, so that two log groups could be archived per tape.

With multiplexed groups of online redo logs, all members of the same 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: The default size of online redo log files is operating system-dependent; for more details see your operating system-specific Oracle documentation.

Choosing the Number of Online Redo Log Files

The best way to determine the appropriate number of online redo log files for a database instance is to test different configurations. The optimum configuration has

the fewest groups possible without hampering LGWR's 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 if the current online redo log configuration is satisfactory is to examine the contents of the LGWR trace file and the database's 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 online redo log files before setting up or altering the configuration of an instance's online redo log. The following parameters limit the number of online 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 online redo log files per 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; thus, it is important to consider this limit *before* creating a database. If MAXLOGFILES is not specified for the CREATE DATABASE statement, Oracle uses an operating system default value.
- The LOG_FILES initialization parameter (in the parameter file) can temporarily decrease the maximum number of groups of online redo log files for the duration of the current instance. Nevertheless, LOG_FILES cannot override MAXLOGFILES to increase the limit. If LOG_FILES is not set in the database's parameter file, Oracle uses an operating system-specific default value.
- The MAXLOGMEMBERS parameter used in the CREATE DATABASE statement determines the maximum number of members per group. As with MAXLOGFILES, the only way to override this upper limit is to re-create the database or control file; thus, it is important to consider this limit *before* creating a database. If no MAXLOGMEMBERS parameter is specified for the CREATE DATABASE statement, Oracle uses an operating system default value.

See Also: For the default and legal values of the MAXLOGFILES and MAXLOGMEMBERS parameters, and the LOG_FILES initialization parameter, see your operating system-specific Oracle documentation.

Creating Online Redo Log Groups and Members

You can create groups and members of online redo log files during or after database creation. Plan the online redo log of a database and create all required groups and members of online redo log files during database creation. To create new online redo log groups and members, you must have the ALTER DATABASE system privilege.

In some cases, you might need to create additional groups or members of online redo log files. For example, adding groups to an online redo log can correct redo log group availability problems. A database can have up to MAXLOGFILES groups.

Creating Online Redo Log Groups

To create a new group of online redo log files, use the SQL statement ALTER DATABASE with the ADD LOGFILE parameter.

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 file is created in the default directory of the database server, which is operating system-dependent. To reuse an existing operating system file, you do not have to indicate the file size.

Using the ALTER DATABASE statement with the ADD LOGFILE option, you can specify the number that identifies the group with the GROUP option:

```
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 space in the control files of the database.

Creating Online Redo Log Members

In some cases, you might not need to create a complete group of online redo log files; the group may 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 online redo log members for an existing group, use the SQL statement ALTER DATABASE with the ADD LOG MEMBER parameter.

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 command, you can alternatively identify the target group by specifying all of the other members of the group in the TO parameter, 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 the default directory of the database server.

Renaming and Relocating Online Redo Log Members

You can rename online redo log members to change their locations. This procedure is necessary, for example, if the disk currently used for some online redo log files is going to be removed, or if datafiles and a number of online redo log files are stored on the same disk and should be separated to reduce contention.

To rename online 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 renaming any online redo log members, ensure that the new online redo log files already exist.

WARNING: The following steps only modify the internal file pointers in a database's control files; they do not physically rename or create any operating system files. Use your computer's operating system to copy the existing online redo log files to the new location.

To Rename Online Redo Log Members

1. Back up the database.

   Before making any structural changes to a database, such as renaming or relocating online redo log members, completely back up the database (including the control file) in case you experience any problems while performing this operation.

2. Copy the online redo log files to the new location.

   Operating system files, such as online redo log members, must be copied using the appropriate operating system commands. See your operating system manual for more information about copying files.

   Note: You can execute an operating system command to copy a file without exiting SQL*Plus by using the HOST command.

3. Rename the online redo log members.

   Use the ALTER DATABASE statement with the RENAME FILE clause to rename the database's online redo log files.

4. Open the database for normal operation.

   The online redo log alterations take effect the next time that the database is opened. Opening the database may require shutting down the current instance (if the database was previously opened by the current instance) or just opening the database using the current instance.

5. Back up the control file.

   As a precaution, after renaming or relocating a set of online redo log files, immediately back up the database's control file.

The following example renames the online redo log members. However, first assume that:

- The database is currently mounted by, but closed to, the instance.
- The log files are located on two disk: `diska` and `diskb`.
- The online 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 online 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`.

The files `/diska/logs/log1a.rdo` and `/diska/logs/log2a.rdo` on `diska` must be copied to the new files `/diskc/logs/log1c.rdo` and `/diskc/logs/log2c.rdo` on `diskc`.

```
ALTER DATABASE RENAME FILE '/diska/logs/log1a.rdo', '/diska/logs/log2a.rdo'
TO '/diskc/logs/log1c.rdo', '/diskc/logs/log2c.rdo';
```

Dropping Online Redo Log Groups and Members

In some cases, you may want to drop an entire group of online redo log members. For example, you want to reduce the number of groups in an instance's online redo log. In a different case, you may want to drop one or more specific online redo log members. For example, if a disk failure occurs, you may need to drop all the online redo log files on the failed disk so that Oracle does not try to write to the inaccessible files. In other situations, particular online redo log files become unnecessary; for example, a file might be stored in an inappropriate location.

Dropping Log Groups

To drop an online redo log group, you must have the ALTER DATABASE system privilege. Before dropping an online redo log group, consider the following restrictions and precautions:

- An instance requires at least two groups of online redo log files, regardless of the number of members in the groups. (A group is one or more members.)
- You can drop an online redo log group only if it is not the active group. If you need to drop the active group, first force a log switch to occur; see "Forcing Log Switches" on page 6-16.
- Make sure an online redo log group is archived (if archiving is enabled) before dropping it. To see whether this has happened, use the SQL*Plus ARCHIVE LOG statement with the LIST parameter.

Drop an online 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 an online redo log group is dropped from the database, 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 an online redo log group, make sure that the drop completed successfully, and then use the appropriate operating system command to delete the dropped online redo log files.

Dropping Online Redo Log Members

To drop an online redo log member, you must have the ALTER DATABASE system privilege.

Consider the following restrictions and precautions before dropping individual online redo log members:

- It is permissible to drop online redo log files so that a multiplexed online redo log becomes temporarily asymmetric. For example, if you use duplexed groups of online 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 online redo log.
- An instance always requires at least two valid groups of online redo log files, regardless of the number of members in the groups. (A group is 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's status, use the V$LOGFILE view. A redo log file becomes INVALID if Oracle cannot access it. It becomes STALE if Oracle 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 an online redo log member only if it is *not* part of an active 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 an online redo log member belongs is archived (if archiving is enabled) before dropping the member. To see whether this has happened, use the SQL*Plus ARCHIVE LOG command with the LIST parameter.

To drop specific inactive online redo log members, use the SQL 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 an online 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 an online redo log file, make sure that the drop completed successfully, and then use the appropriate operating system command to delete the dropped online redo log file.

See Also: For information on dropping a member of an active group, see "Forcing Log Switches" on page 6-16.

For more information about SQL*Plus command syntax, see the *SQL*Plus User's Guide and Reference*.

Forcing Log Switches

A log switch occurs when LGWR stops writing to one online redo log group and starts writing to another. By default, a log switch occurs automatically when the current online redo log file group fills.

You can force a log switch to make the currently active group inactive and available for online 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 online redo log files that take a long time to fill.

To force a log switch, you must have the ALTER SYSTEM privilege. To force a log switch, use either the SQL statement ALTER SYSTEM with the SWITCH LOGFILE option.

The following statement forces a log switch:

```
ALTER SYSTEM SWITCH LOGFILE;
```

See Also: For information on forcing log switches with the Oracle Parallel Server, see *Oracle8i Parallel Server Concepts and Administration*.

Verifying Blocks in Redo Log Files

You can configure Oracle to use checksums to verify blocks in the redo log files. Set the initialization parameter LOG_BLOCK_CHECKSUM to TRUE to enable redo log block checking. The default value of LOG_BLOCK_CHECKSUM is FALSE.

If you enable redo log block checking, Oracle computes a *checksum* for each redo log block written to the current log. Oracle writes the checksums in the header of the block.

Oracle uses the checksum to detect corruption in a redo log block. Oracle tries to verify the redo log block when it writes the block to an archive log file and when the block is read from an archived log during recovery.

If Oracle detects a corruption in a redo log block while trying to archive it, the system tries to read the block from another member in the group. If the block is corrupted in all members the redo log group, then archiving cannot proceed.

Clearing an Online Redo Log File

If you have enabled redo log block checking, Oracle verifies each block before archiving it. If a particular redo log block is corrupted in all members of a group, archiving stops. Eventually all the redo logs become filled and database activity is halted until archiving can resume.

In this situation, use the SQL statement ALTER DATABASE ... CLEAR LOGFILE to clear the corrupted redo logs and avoid archiving them. The cleared redo logs are available for use even though they were not archived.

The following statement clears the log files in redo log group number 3:

```
ALTER DATABASE CLEAR UNARCHIVED LOGFILE GROUP 3;
```

Restrictions

You can clear a redo log file whether it is archived or not. When it is not archived, however, you must include the keyword UNARCHIVED in your ALTER DATABASE CLEAR LOGFILE statement.

If you clear a log file that is needed for recovery of a backup, then you can no longer recover from that backup. Oracle 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 clause UNRECOVERABLE DATAFILE 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.

See Also: For a complete description of the ALTER DATABASE statement, see the *Oracle8i SQL Reference*.

Listing Information about the Online Redo Log

Use the VLOG, VLOGFILE, and V$THREAD views to see information about the online redo log of a database; the V$THREAD view is of particular interest for Parallel Server administrators.

The following query returns information about the online redo log of a database used without the Parallel Server:

```
SELECT group#, bytes, members FROM sys.v$log;

GROUP#        BYTES        MEMBERS
----------    ----------   ----------
         1      81920       2
         2      81920       2
```

To see the names of all of the member of a group, use a query similar to the following:

```
SELECT * FROM sys.v$logfile
   WHERE group# = 2;

GROUP#         STATUS       MEMBER
----------    ----------  ---------
         2                  LOG2A
         2    STALE         LOG2B
         2                  LOG2C
```

If STATUS is blank for a member, then the file is in use.

7

Managing Archived Redo Logs

This chapter describes how to archive redo data. It includes the following topics:

- What Is the Archived Redo Log?
- Choosing Between NOARCHIVELOG and ARCHIVELOG Mode
- Turning Archiving On and Off
- Specifying the Archive Destination
- Specifying the Mode of Log Transmission
- Managing Archive Destination Failure
- Tuning Archive Performance
- Displaying Archived Redo Log Information
- Using LogMiner to Analyze Online and Archived Redo Logs

See Also: If you are using Oracle with the Parallel Server, see *Oracle8i Parallel Server Concepts and Administration* for additional information about archiving in the OPS environment.

What Is the Archived Redo Log?

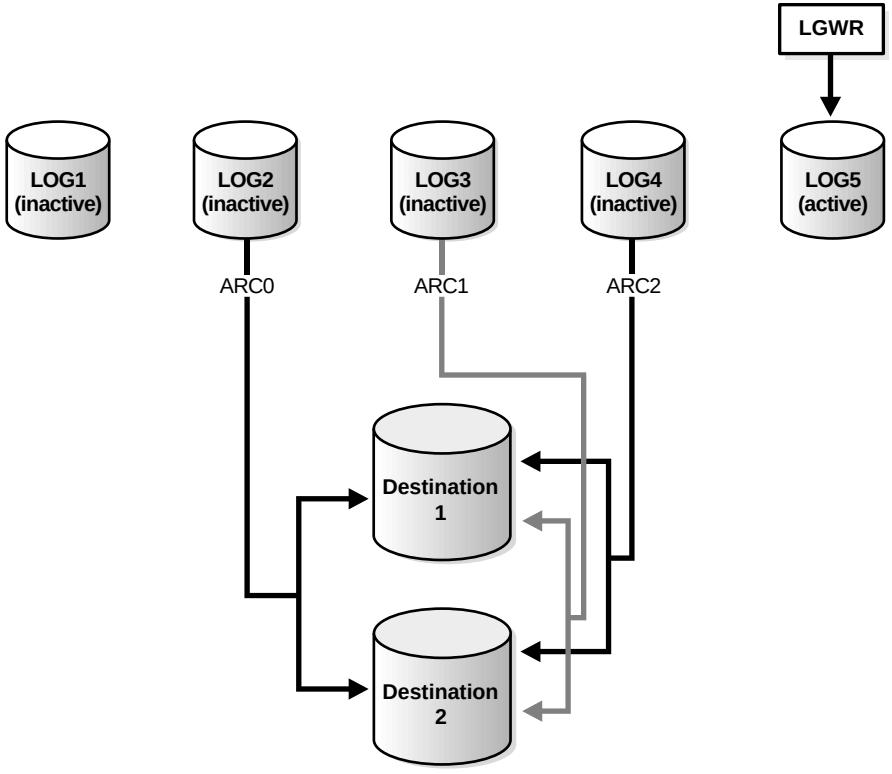
Oracle allows you to save filled groups of online redo log files, known as *archived redo logs*, to one or more offline destinations. *Archiving* is the process of turning online redo logs into archived redo logs. The background process ARC*n* automates archiving operations. You can use archived logs to:

- Recover a database.
- Update a standby database.
- Gain information about the history of a database via the LogMiner utility.

An archived redo log file is a copy of one of the identical filled members of an online redo log group: it includes the redo entries present in the identical members of a group and also preserves the group's unique log sequence number. For example, if you are multiplexing your online redo logs, and if Group 1 contains member files A_LOG1 and B_LOG1, then ARC*n* will archive one of these identical members. Should A_LOG1 become corrupted, then ARC*n* can still archive the identical B_LOG1.

If you enable archiving, LGWR is not allowed to re-use and hence overwrite an online redo log group until it has been archived. Therefore, the archived redo log contains a copy of every group created since you enabled archiving. Figure 7–1 shows how ARC*n* archives redo logs:

Figure 7–1 Archival of online redo logs



WARNING: Oracle recommends that you do not copy a current online log. If you do, and then restore that copy, the copy will appear at the end of the redo thread. Since additional redo may have been generated in the thread, when you attempt to execute recovery by supplying the redo log copy, recovery will erroneously detect the end of the redo thread and prematurely terminate, possibly corrupting the database. The best way to back up the contents of the current online log is always to archive it, then back up the archived log.

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 includes the following topics:

- Running a Database in NOARCHIVELOG Mode
- Running a Database in ARCHIVELOG Mode

Running a Database in NOARCHIVELOG Mode

When you run your database in NOARCHIVELOG mode, you disable the archiving of the online redo log. The database's 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.

The choice of whether to enable the archiving of filled groups of online 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. Note that the archiving of filled online redo log files can require you to perform extra administrative operations.

NOARCHIVELOG mode protects a database only from instance failure, but not from media failure. Only the most recent changes made to the database, which are stored in the groups of the online redo log, are available for instance recovery. In other words, if you are using NOARCHIVELOG mode, you can only *restore* (not recover) the database to the point of the most recent full database backup. You cannot recover subsequent transactions.

Also, in NOARCHIVELOG mode you cannot perform online tablespace backups. Furthermore, you cannot use online tablespace backups previously taken while the database operated in ARCHIVELOG mode. You can only use whole database backups taken while the database is closed to restore a database operating in NOARCHIVELOG mode. 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 online redo log. The database control file indicates that a group of filled online redo

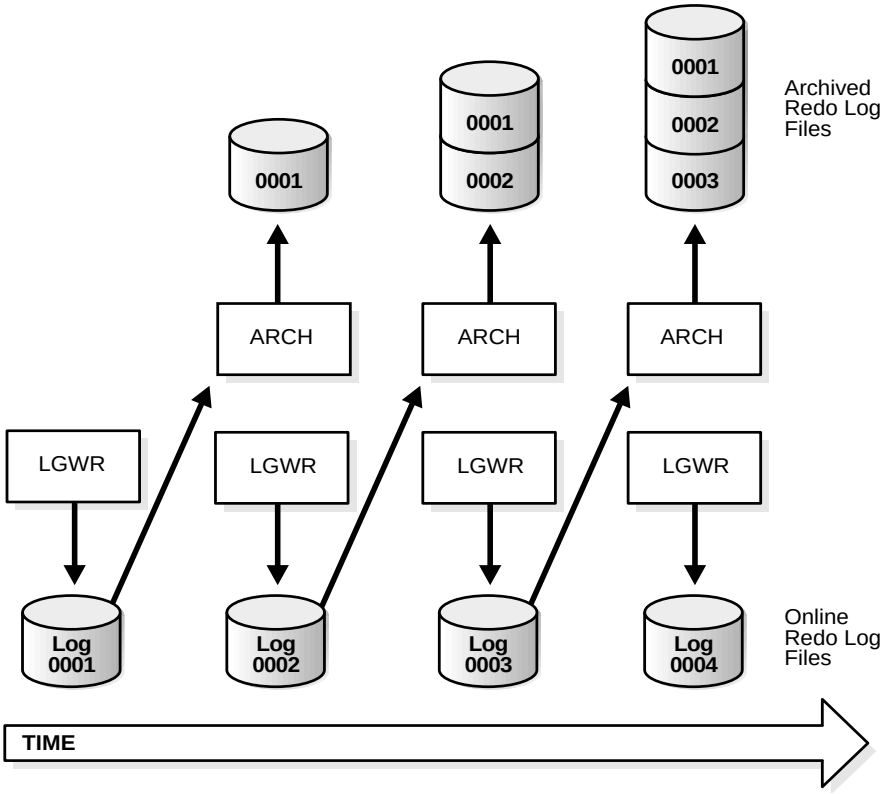
log files cannot be used by LGWR until the group is archived. A filled group is immediately available for archiving after a redo log switch occurs.

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.
- You can use a backup taken while the database is open and in normal system use if you keep an archived log.
- You can keep a standby database current with its original database by continually applying the original's archived redo logs to the standby.

Decide how you plan to archive filled groups of the online redo log. You can configure an instance to archive filled online redo log files automatically, or you can archive manually. For convenience and efficiency, automatic archiving is usually best. Figure 7–2 illustrate how the process archiving the filled groups (ARC*n* in this illustration) generates the database's online redo log.

Figure 7–2 Online Redo Log File Use in ARCHIVELOG Mode



Distributed Database Recovery If *all* databases in a distributed database operate in ARCHIVELOG mode, you can perform coordinated distributed database recovery. If *any* database in a distributed database uses NOARCHIVELOG mode, however, 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.

See Also: You can also configure Oracle to verify redo log blocks when they are archived. For more information, see "Verifying Blocks in Redo Log Files" on page 6-16.

Turning Archiving On and Off

This section describes aspects of archiving, and includes the following topics:

- Setting the Initial Database Archiving Mode
- Changing the Database Archiving Mode
- Enabling Automatic Archiving
- Disabling Automatic Archiving
- Performing Manual Archiving

See Also: If a database is automatically created during Oracle installation, the initial archiving mode of the database is operating system specific. See your operating system-specific Oracle documentation.

Setting the Initial Database Archiving Mode

You set a database's 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 then. After creating the database, decide whether to change from the initial archiving mode.

Changing the Database Archiving Mode

To switch a database's archiving mode between NOARCHIVELOG and ARCHIVELOG mode, use the SQL statement ALTER DATABASE with the ARCHIVELOG or NOARCHIVELOG option. The following statement switches the database's archiving mode from NOARCHIVELOG to ARCHIVELOG:

```
ALTER DATABASE ARCHIVELOG;
```

Before switching the database's archiving mode, perform the following operations:

1. Shut down the database instance.

   An open database must be closed and dismounted and any associated instances shut down before you can switch the database's archiving mode. You cannot disable archiving 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.

3. Start a new instance and mount but do not open the database.

   To enable or disable archiving, the database must be mounted but not open.

   Note: If you are using the Oracle Parallel Server, you must mount the database exclusively, using one instance, to switch the database's archiving mode.

4. Switch the database's archiving mode.

After using the ALTER DATABASE command to switch a database's archiving mode, open the database for normal operation. If you switched to ARCHIVELOG mode, you should also set the archiving options—decide whether to enable Oracle to archive groups of online redo log files automatically as they fill.

If you want to archive filled groups, you may have to execute some additional steps, depending on your operating system; see your O/S-specific Oracle documentation for details for your system.

See Also: See *Oracle8i Parallel Server Concepts and Administration* for more information about switching the archiving mode when using the Oracle Parallel Server.

Enabling Automatic Archiving

If your operating system permits, you can enable automatic archiving of the online redo log. Under this option, no action is required to copy a group after it fills; Oracle automatically archives it. For this convenience alone, automatic archiving is the method of choice for archiving.

You can enable automatic archiving before or after instance startup. To enable automatic archiving after instance startup, you must be connected to Oracle with administrator privileges.

WARNING: Oracle does not automatically archive log files unless the database is also in ARCHIVELOG mode.

See Also: Always specify an archived redo log destination and filename format when enabling automatic archiving; see "Specifying Archive Destinations" on page 7-11. If automatic archiving is enabled, you can still perform manual archiving; see "Performing Manual Archiving" on page 7-10.

Enabling Automatic Archiving at Instance Startup

To enable automatic archiving of filled groups each time an instance is started, include the initialization parameter LOG_ARCHIVE_START parameter in the database's parameter file and set it to TRUE:

```
LOG_ARCHIVE_START=TRUE
```

The new value takes effect the next time you start the database.

Enabling Automatic Archiving After Instance Startup

To enable automatic archiving of filled online redo log groups without shutting down the current instance, use the SQL statement ALTER SYSTEM with the ARCHIVE LOG START parameter; you can optionally include the archiving destination.

```
ALTER SYSTEM ARCHIVE LOG START;
```

If you use the ALTER SYSTEM method, you do not need to shut down the instance to enable automatic archiving. If an instance is shut down and restarted after automatic archiving is enabled, however, the instance is reinitialized using the settings of the parameter file, which may or may not enable automatic archiving.

Disabling Automatic Archiving

You can disable automatic archiving of the online redo log groups at any time. Once you disable automatic archiving, however, you must manually archive groups of online redo log files in a timely fashion. If you run a database in ARCHIVELOG mode and disable automatic archiving, and if all groups of online redo log files are filled but not archived, then LGWR cannot reuse any inactive groups of online redo log groups to continue writing redo log entries. Therefore, database operation is temporarily suspended until you perform the necessary archiving.

You can disable automatic archiving at or after instance startup. To disable automatic archiving after instance startup, you must be connected with administrator privilege and have the ALTER SYSTEM privilege.

Disabling Automatic Archiving at Instance Startup

To disable the automatic archiving of filled online redo log groups each time a database instance is started, set the LOG_ARCHIVE_START parameter of a database's parameter file to FALSE:

```
LOG_ARCHIVE_START=FALSE
```

The new value takes effect the next time the database is started.

Disabling Automatic Archiving after Instance Startup

To disable the automatic archiving of filled online redo log groups without shutting down the current instance, use the SQL statement ALTER SYSTEM with the ARCHIVE LOG STOP parameter. The following statement stops archiving:

```
ALTER SYSTEM ARCHIVE LOG STOP;
```

If ARC*n* is archiving a redo log group when you attempt to disable automatic archiving, ARC*n* finishes archiving the current group, but does not begin archiving the next filled online redo log group.

The instance does not have to be shut down to disable automatic archiving. If an instance is shut down and restarted after automatic archiving is disabled, however, the instance is reinitialized using the settings of the parameter file, which may or may not enable automatic archiving.

Performing Manual Archiving

If you operate your database in ARCHIVELOG mode, then you must archive inactive groups of filled online redo log files. You can manually archive groups of the online redo log whether or not automatic archiving is enabled:

- If automatic archiving is not enabled, then you must manually archive groups of filled online redo log files in a timely fashion. If all online redo log groups are filled but not archived, LGWR cannot reuse any inactive groups of online redo log members to continue writing redo log entries. Therefore, database operation is temporarily suspended until the necessary archiving is performed.
- If automatic archiving is enabled, but you want to rearchive an inactive group of filled online redo log members to another location, you can use manual archiving. Note that the instance can decide to reuse the redo log group before you have finished manually archiving, and thereby overwrite the files; if this happens, Oracle will put an error message in the ALERT file.

To archive a filled online redo log group manually, connect with administrator privileges. Use the SQL statement ALTER SYSTEM with the ARCHIVE LOG clause to manually archive filled online redo log files. The following statement archives all unarchived log files:

```
ALTER SYSTEM ARCHIVE LOG ALL;
```

See Also: With both manual or automatic archiving, you need to specify a thread only when you are using the Oracle Parallel Server. See *Oracle8i Parallel Server Concepts and Administration* for more information.

Specifying the Archive Destination

When archiving redo logs, determine the destination to which you will archive. You should familiarize yourself with the various destination states as well as the practice of using fixed views to access archive information.

Specifying Archive Destinations

You must decide whether to make a *single* destination for the logs or *multiplex* them, i.e., archive the logs to more than one location.

Specify the number of locations for your primary database archived logs by setting the following initialization parameters:

Parameter	Host	Example
LOG_ARCHIVE_DEST_*n* (where *n* is an integer from 1 to 5)	Remote or local	LOG_ARCHIVE_DEST_1 = 'LOCATION = /disk1/arc' LOG_ARCHIVE_DEST_2 = 'SERVICE = standby1'
LOG_ARCHIVE_DEST and LOG_ARCHIVE_DUPLEX_DEST	Local only	LOG_ARCHIVE_DEST = /oracle/arc LOG_ARCHIVE_DUPLEX_DEST = /bak

The first method is to use the LOG_ARCHIVE_DEST_*n* parameter (where *n* is an integer from 1 to 5) to specify from one to five different destinations for archival. Each numerically-suffixed parameter uniquely identifies an individual destination, For example, LOG_ARCHIVE_DEST_1, LOG_ARCHIVE_DEST_2, and so on.

Specify the location for LOG_ARCHIVE_DEST_*n* using these keywords:

Keyword	Indicates	Example
LOCATION	A local filesystem location.	LOG_ARCHIVE_DEST_1= 'LOCATION=/arc'
SERVICE	Remote archival via Net8 service name.	LOG_ARCHIVE_DEST_2 = 'SERVICE=standby1'

If you use the LOCATION keyword, specify a valid pathname for your operating system. If you specify SERVICE, Oracle 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. Note that the service name must have an associated database SID, so that Oracle correctly updates the log history of the control file for the standby database.

The second method, which allows you to specify a maximum of two locations, is to use the LOG_ARCHIVE_DEST parameter to specify a *primary* archive destination and the LOG_ARCHIVE_DUPLEX_DEST to determine an optional *secondary* location. Whenever Oracle archives a redo log, it archives it to every destination specified by either set of parameters.

To Set the Destination for Archived Redo Logs Using LOG_ARCHIVE_DEST_*n*:

1. Use SQL*Plus to shut down the database.

   ```
   SHUTDOWN IMMEDIATE;
   ```

2. Edit the LOG_ARCHIVE_DEST_*n* parameter to specify from one to five archiving locations. The LOCATION keyword specifies an O/S-specific pathname. 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. Edit the LOG_ARCHIVE_FORMAT parameter, using `%s` to include the log sequence number as part of the filename and `%t` to include the thread number. Use capital letters (`%S` and `%T`) to pad the filename to the left with zeroes. For example, enter:

   ```
   LOG_ARCHIVE_FORMAT = arch%s.arc
   ```

   For example, the above settings will generate archived logs as follows for log sequence numbers 100, 101, and 102:

   ```
   /disk1/archive/arch100.arc, /disk1/archive/arch101.arc, /disk1/archive/arch102.arc
   /disk2/archive/arch100.arc, /disk2/archive/arch101.arc, /disk2/archive/arch102.arc
   /disk3/archive/arch100.arc, /disk3/archive/arch101.arc, /disk3/archive/arch102.arc
   ```

To Set the Destination for Archived Redo Logs Using LOG_ARCHIVE_DEST and LOG_ARCHIVE_DUPLEX_DEST:

1. Use SQL*Plus to shut down the database.

```
SHUTDOWN IMMEDIATE;
```

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 command). For example, enter:

```
LOG_ARCHIVE_DEST = '/disk1/archive'
LOG_ARCHIVE_DUPLEX_DEST = '/disk2/archive'
```

3. Edit the LOG_ARCHIVE_FORMAT parameter, using %S to include the log sequence number as part of the filename and %t to include the thread number. Use capital letters (%S and %T) to pad the filename to the left with zeroes. For example, enter:

```
LOG_ARCHIVE_FORMAT = arch_%t_%s.arc
```

   For example, the above settings will generate archived logs as follows for log sequence numbers 100 and 101 in thread 1:

```
/disk1/archive/arch_1_100.arc, /disk1/archive/arch_1_101.arc
/disk2/archive/arch_1_100.arc, /disk2/archive/arch_1_100.arc
```

See Also: For more information about archiving to standby databases, see *Oracle8i Backup and Recovery Guide*.

Understanding Archive Destination States

The LOG_ARCHIVE_DEST_STATE_*n* (where *n* is an integer from 1 to 5) parameter identifies the status of the specified destination. The destination parameters can have two values: ENABLE and DEFER. ENABLE indicates that Oracle can use the destination, whereas DEFER indicates that it should not.

Each archive destination has three variable characteristics:

- *Valid/Invalid*, which indicates whether the disk location or service name information is specified.
- *Enabled/Disabled*, which indicates whether Oracle should use the destination information.

- *Active/Inactive*, which indicates whether there was a problem accessing the destination.

Several destination states are possible. Obtain the current destination status information for each instance by querying the V$ARCHIVE_DEST view. You will access the most recently entered parameter definition—which does not necessarily contain the complete archive destination data.

The status information that appears in the view is shown in Table 7–1:

Table 7–1 Destination States

VALID	ENABLED	ACTIVE	State	Meaning
FALSE	N / A	N / A	INACTIVE	The user has not provided or has deleted the destination information.
TRUE	TRUE	TRUE	VALID	The user has properly initialized the destination, which is available for archiving.
TRUE	TRUE	FALSE	ERROR	An error occurred creating or writing to the destination file; refer to error data.
TRUE	FALSE	TRUE	DEFERRED	The user manually and temporarily disabled the destination.
TRUE	FALSE	FALSE	DISABLED	The user manually and temporarily disabled the destination following an error; refer to error data.
N / A	N / A	N / A	BAD PARAM	A parameter error occurred; refer to error data. Usually this state is only seen when LOG_ARCHIVE_START is not set.

See Also: For detailed information about V$ARCHIVE_DEST as well as the archive destination parameters, see the *Oracle8i Reference*.

Specifying the Mode of Log Transmission

There are two modes of transmitting archived logs to their destination: *normal archiving transmission* and *standby transmission* mode. Normal transmission involves

transmitting files to a local disk. Standby transmission involves transmitting files via 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, since in this configuration archiving does not contend with other files required by the instance and completes quickly so the group can become available to LGWR. Specify the destination with either the LOG_ARCHIVE_DEST_*n* or LOG_ARCHIVE_DEST parameters.

Ideally, you should permanently move archived redo log files and corresponding database backups from the local disk to inexpensive offline storage media such as tape. Because a primary value of archived logs is database recovery, 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.

> **WARNING: 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 in sync 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 by means of a *remote file server* (RFS) process.

Each ARC*n* process creates a corresponding RFS for each standby destination. For example, if three ARC*n* processes are archiving to two standby databases, then Oracle establishes six RFS connections.

You can transmit archived logs through a network to a remote location by using Net8. Indicate a remote archival by specifying a Net8 *service name* as an attribute of the destination. Oracle then translates the service name, which you set by means of

the SERVICE_NAME parameter, through the `tnsnames.ora` file to a *connect descriptor*. The descriptor contains the information necessary for connecting to the remote database. Note that the service name must have an associated database SID, so that Oracle 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's 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: For a detailed description of standby databases, see the relevant chapter in the *Oracle8i Backup and Recovery Guide*.

For information about Net8, see the *Net8 Administrator's Guide*.

Managing Archive Destination Failure

Sometimes archive destinations can fail, which can obviously cause problems when you operate in automatic archiving mode. To minimize the problems associated with destination failure, Oracle8*i* allows you to specify:

- The minimum number of destinations to which Oracle must successfully archive.
- When and how often ARC*n* attempts to re-archive to a failed destination.

Specifying the Minimum Number of Successful Destinations

The optional parameter LOG_ARCHIVE_MIN_SUCCEED_DEST=*n* (where *n* is an integer from 1 to 5) determines the minimum number of destinations to which Oracle must successfully archive a redo log group before it can reuse online log files. The default value is 1.

Specifying Mandatory and Optional Destinations

Using the LOG_ARCHIVE_DEST_*n* parameter, you can specify whether a destination has the attributes OPTIONAL (default) or MANDATORY. The LOG_ARCHIVE_MIN_SUCCEED_DEST=*n* parameter uses all MANDATORY destinations plus some number of OPTIONAL non-standby destinations to determine whether LGWR can over-write the online log.

When determining whether how to set your parameters, note that:

- Not specifying MANDATORY 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 using LOG_ARCHIVE_MIN_SUCCEED_DEST=*n* at least one local destination will *operationally* be treated as MANDATORY, since the minimum value for LOG_ARCHIVE_MIN_SUCCEED_DEST is 1.
- The failure of any MANDATORY destination, including a MANDATORY standby destination, makes the LOG_ARCHIVE_MIN_SUCCEED_DEST parameter irrelevant.
- The LOG_ARCHIVE_MIN_SUCCEED_DEST value cannot be greater than the number of destinations, nor greater than the number of MANDATORY destinations plus the number of OPTIONAL local destinations.
- If you DEFER a MANDATORY destination, and Oracle 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 wish, you can also determine whether destinations are mandatory or optional by using the LOG_ARCHIVE_DEST and LOG_ARCHIVE_DUPLEX_DEST parameters. Note the following rules:

- Any destination declared via LOG_ARCHIVE_DEST is mandatory.

- Any destination declared via LOG_ARCHIVE_DUPLEX_DEST is optional if LOG_ARCHIVE_MIN_SUCCEED_DEST = 1 and mandatory if LOG_ARCHIVE_MIN_SUCCEED_DEST = 2.

Sample Scenarios

You can see the relationship between the LOG_ARCHIVE_DEST_*n* and LOG_ARCHIVE_MIN_SUCCEED_DEST parameters most easily through sample scenarios. In example 1, 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 our example.

Table 7–2 LOG_ARCHIVE_MIN_SUCCEED_DEST Values for Example 1

Value	Meaning
1	Oracle can reuse log files only if at least one of the OPTIONAL destinations succeeds.
2	Oracle can reuse log files only if at least two of the OPTIONAL destinations succeed.
3	Oracle can reuse log files only if all of the OPTIONAL destinations succeed.
4	ERROR: The value is greater than the number of destinations.
5	ERROR: The value is greater than the number of destinations.

This example shows that even though you do not explicitly set any of your destinations to MANDATORY using the LOG_ARCHIVE_DEST_*n* parameter, Oracle must successfully archive to these locations when LOG_ARCHIVE_MIN_SUCCEED_DEST is set to 1, 2, or 3.

In example 2, 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 Example 2

Value	Meaning
1	Oracle ignores the value and uses the number of MANDATORY destinations (in this example, 2).
2	Oracle can reuse log files even if no OPTIONAL destination succeeds.
3	Oracle can reuse logs only if at least one OPTIONAL destination succeeds.
4	Oracle can reuse logs only if both OPTIONAL destinations succeed.
5	ERROR: The value is greater than the number of destinations.

This example shows that Oracle 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.

See Also: For additional information about LOG_ARCHIVE_MIN_SUCCEED_DEST=*n* or any other parameters that relate to archiving, see the *Oracle8i Reference*.

Re-Archiving to a Failed Destination

Use the REOPEN attribute of the LOG_ARCHIVE_DEST_*n* parameter to determine whether and when ARC*n* attempts to re-archive 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 option, in other words, 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 a limit on the number of attempts to reconnect and transfer archived logs. The REOPEN attempt either succeeds or fails, in which case the REOPEN information is reset.

If you specify REOPEN for an OPTIONAL destination, Oracle can overwrite online logs if there is an error. If you specify REOPEN for a MANDATORY destination, Oracle stalls the production database when it cannot successfully archive. This scenario requires you to:

- 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 that:

- 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 a REOPEN time was specified or defaulted, 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.

Tuning Archive Performance

For most databases, ARC*n* has no effect on overall system performance. On some large database sites, however, archiving can have an impact on system performance. On one hand, if ARC*n* works very quickly, overall system performance can be reduced while ARC*n* runs, since CPU cycles are being consumed in archiving. On the other hand, if ARC*n* runs extremely slowly, it has little detrimental effect on system performance, but it takes longer to archive redo log files, and can create a bottleneck if all redo log groups are unavailable because they are waiting to be archived.

Use the following methods to tune archiving:

- Specifying Multiple ARCn Processes
- Setting Archive Buffer Parameters

See Also: For more information about tuning a database, see *Oracle8i Tuning*.

Specifying Multiple ARC*n* Processes

Specify up to ten ARC*n* processes for each database instance. Enable the multiple processing feature at startup or at runtime by setting the parameter LOG_ARCHIVE_MAX_PROCESSES=*n* (where *n* is any integer from 1 to 10). By default, the parameter is set to 0.

Because LGWR automatically increases the number of ARC*n* processes should the current number be insufficient to handle the current workload, the parameter is

intended to allow you to specify the *initial* number of ARC*n* processes or to increase or decrease the current number.

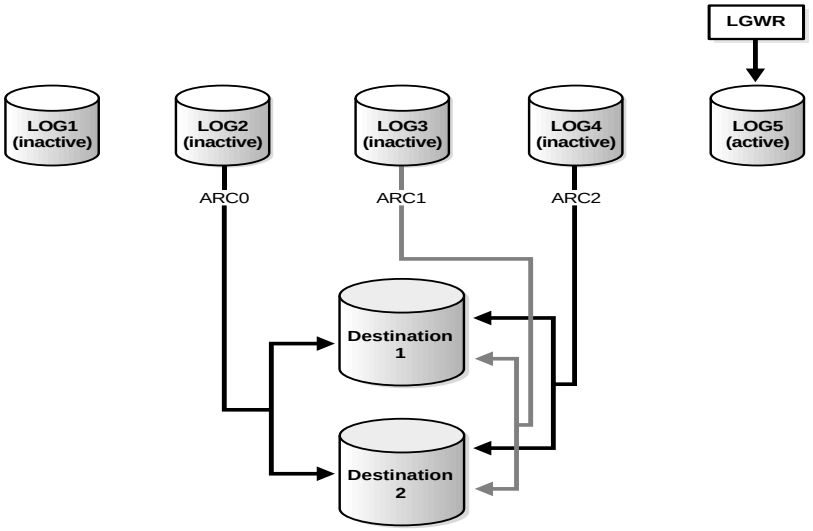
Creating multiple processes is especially useful when you:

- Use more than two online redo logs.
- Archive to more than one destination.

Multiple ARC*n* processing prevents the bottleneck that occurs when LGWR switches through the multiple online redo logs faster than a single ARC*n* process can write inactive logs to multiple destinations. Note that each ARC*n* process works on only one inactive log at a time, but must archive to each specified destination.

For example, if you maintain five online redo log files, then you may decide to start the instance using three ARC*n* processes. As LGWR actively writes to one of the log files, the ARC*n* processes can simultaneously archive up to three of the inactive log files to various destinations. As Figure illustrates, each instance of ARC*n* assumes responsibility for a single log file and archives it to all of the defined destinations.

Figure 7–3 Using Multiple Arch Processes



Setting Archive Buffer Parameters

This section describes aspects of using the archive buffer initialization parameters for tuning, and includes the following topics:

- Minimizing the Impact on System Performance
- Improving Archiving Speed

You can tune archiving to cause it to run either as slowly as possible without being a bottleneck or as quickly as possible without reducing system performance substantially. To do so, adjust the values of the initialization parameters LOG_ARCHIVE_BUFFERS (the number of buffers allocated to archiving) and LOG_ARCHIVE_BUFFER_SIZE (the size of each such buffer).

Note: When you change the value of LOG_ARCHIVE_BUFFERS or LOG_ARCHIVE_BUFFER_SIZE, the new value takes effect the next time you start the instance.

Minimizing the Impact on System Performance

To make ARC*n* work as slowly as possible without forcing the system to wait for redo logs, begin by setting the number of archive buffers (LOG_ARCHIVE_BUFFERS) to 1 and the size of each buffer (LOG_ARCHIVE_BUFFER_SIZE) to the maximum possible.

If the performance of the system drops significantly while ARC*n* is working, make the value of LOG_ARCHIVE_BUFFER_SIZE lower until system performance is no longer reduced when ARC*n* runs.

Note: If you want to set archiving to be very slow, but find that Oracle frequently has to wait for redo log files to be archived before they can be reused, you can create additional redo log file groups. Adding groups can ensure that a group is always available for Oracle to use.

Improving Archiving Speed

To improve archiving performance, use multiple archive buffers to force the ARC*n* process or processes to read the archive log at the same time that they write the output log. You can set LOG_ARCHIVE_BUFFERS to 2, but for a very fast tape drive you may want to set it to 3 or more. Then,set the size of the archive buffers to a moderate number, and increase it until archiving is as fast as you want it to be without impairing system performance.

See Also: This maximum is operating system dependent; see your operating system-specific Oracle documentation. For more information about the LOG_ARCHIVE parameters, see the *Oracle8i Reference*.

Displaying Archived Redo Log Information

There are several fixed views that contain useful information about archived redo logs.

Fixed View	Description
V$DATABASE	Identifies whether the database is in ARCHIVELOG or NOARCHIVELOG mode.

Fixed View	Description
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.
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 online 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 online 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
```

The SQL statement ARCHIVE LOG LIST also shows archiving information for the connected instance:

```
ARCHIVE LOG LIST;
```

```
Database log mode                  ARCHIVELOG
Automatic archival                 ENABLED
Archive destination                destination
Oldest online log sequence         30
Next log sequence to archive       32
Current log sequence number        33
```

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 destination of the archived redo log (operating system specific).
- The oldest filled online redo log group has a sequence number of 30.
- The next filled online redo log group to archive has a sequence number of 32.
- The current online redo log file has a sequence number of 33.

You must archive all redo log groups with a sequence number equal to or greater than the *Next log sequence to archive*, yet less than the *Current log sequence number*. For example, the display above indicates that the online redo log group with sequence number 32 needs to be archived.

See Also: For more information on the data dictionary views, see the *Oracle8i Reference*.

Using LogMiner to Analyze Online and Archived Redo Logs

The Oracle utility LogMiner allows you to read information contained in online and archived redo logs based on selection criteria. LogMiner's fully relational SQL interface provides direct access to a complete historical view of a database—without forcing you to restore archived redo log files.

This section contains the following topics:

- Using LogMiner: Scenarios

How Can You Use LogMiner?

LogMiner is especially useful for identifying and undoing logical corruption. LogMiner processes redo log files, translating their contents into SQL statements that represent the logical operations performed to the database. The V$LOGMNR_CONTENTS view then lists the reconstructed SQL statements that represent the original operations (SQL_REDO column) and the corresponding SQL statement to undo the operations (SQL_UNDO column). Apply the SQL_UNDO statements to roll back the original changes to the database.

Furthermore, you can use the V$LOGMNR_CONTENTS view to:

- Determine when a logical corruption to the database may have begun, pinpointing the time or SCN to which you need to perform incomplete recovery.
- Track changes to a specific table.
- Track changes made by a specific user.
- Map data access patterns.
- Use archived data for tuning and capacity planning.

See Also: For more information about the LogMiner data dictionary views, see the *Oracle8i Reference*.

Restrictions

LogMiner has the following usage and compatibility requirements. LogMiner only:

- Runs in Oracle version 8.1 or later.
- Analyzes redo log files from any version 8.0 or later database that uses the same database character set and runs on the same hardware platform as the analyzing instance.
- Analyzes the contents of the redo log files completely with the aid of a dictionary created by a PL/SQL package. The dictionary allows LogMiner to translate internal object identifiers and data types to object name and external data formats.
- Obtains information about DML operations on conventional tables. It does not support operations on:

- Index-organized tables
- Clustered tables / indexes
- Non-scalar data types
- Chained rows

Creating a Dictionary File

LogMiner runs in an Oracle instance with the database either mounted or unmounted. LogMiner uses a *dictionary file*, which is a special file that indicates the database that created it as well as the time the file was created. The dictionary file is not required, but is recommended.

Without a dictionary file, the equivalent SQL statements will use Oracle internal object IDs for the object name and present column values as hex data. For example, instead of the SQL statement:

```
INSERT INTO emp(name, salary) VALUES ('John Doe', 50000);
```

LogMiner will display:

```
insert into Object#2581(col#1, col#2) values (hextoraw('4a6f686e20446f65'),
hextoraw('c306'));"
```

Create a dictionary file by mounting a database and then extracting dictionary information into an external file. You must create the dictionary file from the same database that generated the log files you want to analyze. Once created, you can use the dictionary file to analyze redo logs.

When creating the dictionary, specify the following:

- DICTIONARY_FILENAME to name the dictionary file.
- DICTIONARY_LOCATION to specify the location of the file.

To Create a Dictionary File on an Oracle8*i* Database:

1. Make sure to specify a directory for use by the PL/SQL procedure by setting the `init.ora` parameter UTL_FILE_DIR. If you do not reference this parameter, the procedure will fail. For example, set the following to use `/oracle/logs`:

   ```
   UTL_FILE_DIR = /oracle/logs
   ```

2. Use SQL*Plus to mount and then open the database whose files you want to analyze. For example, enter:

```
STARTUP
```

3. Execute the PL/SQL procedure DBMS_LOGMNR_D.BUILD. Specify both a filename for the dictionary and a directory pathname for the file. This procedure creates the dictionary file, which you should use to analyze log files. For example, enter the following to create file `dictionary.ora` in `/oracle/logs`:

```
EXECUTE dbms_logmnr_d.build(
dictionary_filename =>'dictionary.ora',
dictionary_location => '/oracle/logs');
```

To Create a Dictionary File on an Oracle8 Database:

Although LogMiner only runs on databases of release 8.1 or higher, you can use it to analyze redo logs from release 8.0 databases.

1. Use an O/S utility to copy the `dbmslogmnrd.sql` script, which is contained in the `$ORACLE_HOME/rdbms/admin` directory on the Oracle8*i* database, to the same directory in the Oracle8 database. For example, enter:

```
% cp /8.1/oracle/rdbms/admin/dbmslogmnrd.sql /8.0/oracle/rdbms/admin/dbmslogmnrd.sql
```

2. Use SQL*Plus to mount and then open the database whose files you want to analyze. For example, enter:

```
STARTUP
```

3. Execute the copied `dbmslogmnrd.sql` script on the 8.0 database to create the DBMS_LOGMNR_D package. For example, enter:

```
@dbmslogmnrd.sql
```

4. Specify a directory for use by the PL/SQL package by setting the `init.ora` parameter UTL_FILE_DIR. If you do not reference this parameter, the procedure will fail. For example, set the following to use `/8.0/oracle/logs`:

```
UTL_FILE_DIR = /8.0/oracle/logs
```

5. Execute the PL/SQL procedure DBMS_LOGMNR_D.BUILD. Specify both a filename for the dictionary and a directory pathname for the file. This procedure creates the dictionary file, which you should use to analyze log files. For example, enter the following to create file `dictionary.ora` in `/8.0/oracle/logs`:

```
EXECUTE dbms_logmnr_d.build(
dictionary_filename =>'dictionary.ora',
dictionary_location => '/8.0/oracle/logs');
```

See Also: For information about DBMS_LOGMNR_D, see the *Oracle8i Supplied Packages Reference.*

Specifying Redo Logs for Analysis

Once you have created a dictionary file, you can begin analyzing redo logs. Your first step is to specify the log files that you want to analyze using the ADD_LOGFILE procedure. Use the following constants:

- NEW to create a new list.
- ADDFILE to add redo logs to a list.
- REMOVEFILE to remove redo logs from the list.

To Use the LogMiner:

1. Use SQL*Plus to start an Oracle instance, with the database either mounted or unmounted. For example, enter:

```
startup
```

2. Create a list of logs by specifying the NEW option when executing the DBMS_LOGMNR.ADD_LOGFILE procedure. For example, enter the following to specify `/oracle/logs/log1.f`:

```
execute dbms_logmnr.add_logfile(
LogFileName => '/oracle/logs/log1.f',
Options => dbms_logmnr.NEW);
```

3. If desired, add more logs by specifying the ADDFILE option. For example, enter the following to add `/oracle/logs/log2.f`:

```
execute dbms_logmnr.add_logfile(
LogFileName => '/oracle/logs/log2.f',
Options => dbms_logmnr.ADDFILE);
```

4. If desired, remove logs by specifying the REMOVEFILE option. For example, enter the following to remove `/oracle/logs/log2.f`:

```
execute dbms_logmnr.add_logfile(
LogFileName => '/oracle/logs/log2.f',
Options => dbms_logmnr.REMOVEFILE);
```

See Also: For information about DBMS_LOGMNR, see the *Oracle8i Supplied Packages Reference.*

Using LogMiner

Once you have created a dictionary file and specified which logs to analyze, you can start LogMiner and begin your analysis. Use the following options to narrow the range of your search at start time:

This option	Specifies
StartScn	The beginning of an SCN range.
EndScn	The termination of an SCN range.
StartTime	The beginning of a time interval.
EndTime	The end of a time interval.
DictFileName	The name of the dictionary file.

Once you have started LogMiner, you can make use of the following data dictionary views for analysis:

This view	Displays information about
V$LOGMNR_DICTIONARY	The dictionary file in use.
V$LOGMNR_PARAMETERS	Current parameter settings for the LogMiner.
V$LOGMNR_FILES	Which redo log files are being analyzed.
V$LOGMNR_CONTENTS	The contents of the redo log files being analyzed.

To Use the LogMiner:

1. Issue the DBMS_LOGMNR.START_LOGMNR procedure to start the LogMiner utility. For example, to start LogMiner using `/oracle/dictionary.ora`, issue:

```
execute dbms_logmnr.start_logmnr(
DictFileName =>'/oracle/dictionary.ora');
```

   Optionally, set the StartTime and EndTime parameters to filter data by time. Note that the procedure expects date values: use the TO_DATE function to specify date and time, as in this example:

```
execute dbms_logmnr.start_logmnr(
DictFileName => '/oracle/dictionary.ora',
StartTime => to_date('01-Jan-98 08:30:00', 'DD-MON-YYYY HH:MI:SS')
EndTime => to_date('01-Jan-1998 08:45:00', 'DD-MON-YYYY HH:MI:SS'));
```

Use the StartScn and EndScn parameters to filter data by SCN, as in this example:

```
execute dbms_logmnr.start_logmnr(
DictFileName => '/oracle/dictionary.ora',
StartScn => 100,
EndScn => 150);
```

2. View the output via the V$LOGMNR_CONTENTS table. LogMiner returns all rows in SCN order, which is the same order applied in media recovery. For example, the following query lists information about operations:

```
SELECT operation, sql_redo FROM v$logmnr_contents;
OPERATION SQL_REDO
--------- ----------------------------------------------------------
INTERNAL
INTERNAL
START     set transaction read write;
UPDATE    update SYS.UNDO$ set NAME = 'RS0', USER# = 1, FILE# = 1, BLOCK# = 2450, SCNBAS =
COMMIT    commit;
START     set transaction read write;
UPDATE    update SYS.UNDO$ set NAME = 'RS0', USER# = 1, FILE# = 1, BLOCK# = 2450, SCNBAS =
COMMIT    commit;
START     set transaction read write;
UPDATE    update SYS.UNDO$ set NAME = 'RS0', USER# = 1, FILE# = 1, BLOCK# = 2450, SCNBAS =
COMMIT    commit;
11 rows selected.
```

See Also: For information about DBMS_LOGMNR, see the *Oracle8i Supplied Packages Reference*.

For more information about the LogMiner data dictionary views, see *Oracle8i Reference*.

Analyzing Archived Redo Log Files from Other Databases You can run LogMiner on an instance of a database while analyzing redo log files from a different database. To analyze archived redo log files from other databases, LogMiner must:

- Access a dictionary file that is both created from the same database as the redo log files and created with the same database character set.
- Run on the same hardware platform that generated the log files, although it does not need to be on the same system.
- Use redo log files that can be applied for recovery from Oracle version 8.0 and later.

Using LogMiner: Scenarios

This section contains the following LogMiner scenarios:

- Tracking a User
- Calculating Table Access Statistics

Tracking a User

In this example, you are interested in seeing all changes to the database in a specific time range by one of yours users: JOEDEVO. You perform this operation in the following steps:

- Step 1: Creating the Dictionary
- Step 2: Adding Logs and Limiting the Search Range
- Step 3: Starting the LogMiner and Analyzing the Data

Step 1: Creating the Dictionary To use the LogMiner to analyze JOEDEVO's data, you must create a dictionary file before starting LogMiner.

You decide to do the following:

- Call the dictionary file `orc1dict.ora`.
- Place the dictionary in directory `/user/local/dbs`.
- Set the initialization parameter UTL_FILE_DIR to `/user/local/dbs`.

```
# Set the initialization parameter UTL_FILE_DIR in the init.ora file
UTL_FILE_DIR = /user/local/dbs

# Start SQL*Plus and then connect to the database
connect system/manager

# Open the database to create the dictionary file
startup

# Create the dictionary file
execute dbms_logmnr_d.build(
dictionary_filename => 'orcldict.ora',
dictionary_location => '/usr/local/dbs');

# The dictionary has been created and can be used later
shutdown;
```

Step 2: Adding Logs and Limiting the Search Range Now that the dictionary is created, you decide to view the changes that happened at a specific time. You do the following:

- Create a list of log files for use and specify log `log1orc1.ora`.
- Add log `log2orc1.ora` to the list.
- Start LogMiner and limit the search to the range between 8:30 a.m. and 8:45 a.m. on January 1, 1998.

```
# Start SQL*Plus, connect as SYSTEM, then start the instance
connect system/manager
startup nomount

# Supply the list of logfiles to the reader. The Options flag is set to indicate this is a
# new list.

execute dbms_logmnr.add_logfile(Options => dbms_logmnr.NEW,
LogFileName => 'log1orc1.ora');

# Add a file to the existing list. The Options flag is clear to indicate that you are
# adding a file to the existing list

execute dbms_logmnr.add_logfile(Options => dbms_logmnr.ADDFILE,
LogFileName => 'log2orc1.ora');
```

Step 3: Starting the LogMiner and Analyzing the Data At this point the V$LOGMNR_CONTENTS table is available for queries. You decide to find all changes made by user JOEDEVO to the salary table. As you discover, JOEDEVO requested two operations: he deleted his old salary and then inserted a new, higher salary. You now have the data necessary to undo this operation (and perhaps to justify firing JOEDEVO!).

```
# Start the LogMiner. Limit the search to the specified time range.
execute dbms_logmnr.start_logmnr(
DictFileName => 'orcldict.ora',
StartTime => to_date('01-Jan-98 08:30:00', 'DD-MON-YYYY HH:MI:SS')
EndTime => to_date('01-Jan-1998 08:45:00', 'DD-MON-YYYY HH:MI:SS'));

SELECT sql_redo, sql_undo FROM v$logmnr_contents
WHERE username = 'JOEDEVO' AND tablename = 'SALARY';

# The following data is displayed (properly formatted)

SQL_REDO                                SQL_UNDO
--------                                --------
delete * from SALARY                    insert into SALARY(NAME,EMPNO, SAL)
```

```
where EMPNO = 12345                    values ('JOEDEVO', 12345,500)
and ROWID = 'AAABOOAABAAEPCABA';

insert into SALARY(NAME, EMPNO, SAL)    delete * from SALARY
values('JOEDEVO',12345,2500)           where EMPNO = 12345
                                        and ROWID = 'AAABOOAABAAEPCABA';
2 rows selected
```

Calculating Table Access Statistics

The redo logs generated by Oracle RDBMS contain the history of all changes made to the database. Mining the redo logs can thus generate a wealth of information that can be used for tuning the database. In this example, you manage a direct marketing database and want to determine how productive the customer contacts have been in generating revenue for a two week period in August.

First, you start LogMiner and specify a range of times:

```
execute dbms_logmnr.start_logmnr(
StartTime => '07-Aug-98',
EndTime => '15-Aug-98',
DictFileName => '/usr/local/dict.ora');
```

Next, you query V$LOGMNR_CONTENTS to determine which tables have been modified in the time range you specified:

```
select seg_owner, seg_name, count(*) as Hits from
V$LOGMNR_CONTENTS where seg_name not like '%$' group by
seg_owner, seg_name;

SEG_OWNER          SEG_NAME          Hits
---------          --------          ----
CUST               ACCOUNT           384
SCOTT              EMP               12
SYS                DONOR             12
UNIV               DONOR             234
UNIV               EXECDONOR         325
UNIV               MEGADONOR         32
```

See Also: For detailed information about V$LOGMNR_CONTENTS or any of the LogMiner views or initialization parameters, see the *Oracle8i Reference*.

For information about DBMS_LOGMNR.ADD_LOGFILE or any other PL/SQL packages, see the *Oracle8i Supplied Packages Reference*.

8

Managing Job Queues

This chapter describes how to use job queues to schedule periodic execution of PL/SQL code, and includes the following topics:

- SNP Background Processes
- Managing Job Queues
- Viewing Job Queue Information

SNP Background Processes

This section describes SNP background processes and their role in managing job queues, and includes the following topics:

- Multiple SNP processes
- Starting up SNP processes

You can schedule routines to be performed periodically using the job queue. A routine is any PL/SQL code. To schedule a job, you submit it to the job queue and specify the frequency at which the job is to be run. You can also alter, disable, or delete jobs you have submitted.

To maximize performance and accommodate many users, a multi-process Oracle system uses some additional processes called *background processes*. Background processes consolidate functions that would otherwise be handled by multiple Oracle programs running for each user process. Background processes asynchronously perform I/O and monitor other Oracle processes to provide increased parallelism for better performance and reliability.

SNP background processes execute job queues. SNP processes periodically wake up and execute any queued jobs that are due to be run. You must have at least one SNP process running to execute your queued jobs in the background.

SNP background processes differ from other Oracle background processes, in that the failure of an SNP process does not cause the instance to fail. If an SNP process fails, Oracle restarts it.

SNP background processes will not execute jobs if the system has been started in restricted mode. However, you can use the ALTER SYSTEM command to turn this behavior on and off as follows:

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

When you ENABLE a restricted session, SNP background processes do not execute jobs; when you DISABLE a restricted session, SNP background processes execute jobs.

See Also: For more information on SNP background processes, see *Oracle8i Concepts.*

Multiple SNP processes

An instance can have up to 36 SNP processes, named SNP0 to SNP9, and SNPA to SNPZ. If an instance has multiple SNP processes, the task of executing queued jobs can be shared across these processes, thus improving performance. Note, however, that each job is run at any point in time by only one process. A single job cannot be shared simultaneously by multiple SNP processes.

Starting up SNP processes

Job queue initialization parameters enable you to control the operation of the SNP background processes. When you set these parameters in the initialization parameter file for an instance, they take effect the next time you start the instance.

Table 8–1 describes the job queue initialization parameters.

Table 8–1 Job Queue Initialization Parameters

Parameter Name	Description	
JOB_QUEUE_PROCESSES	Default:	0
	Range of values:	0...36
	Multiple instances:	can have different values
	Sets the number of SNP background processes per instance.	
JOB_QUEUE_INTERVAL	Default:	60 (seconds)
	Range of values:	1...3600 (seconds)
	Multiple instances:	can have different values
	Sets the interval between wake-ups for the SNP background processes of the instance.	

Managing Job Queues

This section describes the various aspects of managing job queues, and includes the following topics:

- DBMS_JOB Package
- Submitting a Job to the Job Queue
- How Jobs Execute
- Removing a Job from the Job Queue

- Altering a Job
- Broken Jobs
- Forcing a Job to Execute
- Terminating a Job

DBMS_JOB Package

To schedule and manage jobs in the job queue, use the procedures in the DBMS_JOB package. There are no database privileges associated with using job queues. Any user who can execute the job queue procedures can use the job queue. Table 8–2 lists the job queue procedures in the DBMS_JOB package.

Table 8–2 Procedures in the DBMS_JOB Package

Procedure	Description	Described
SUBMIT	Submits a job to the job queue.	on page 8-4
REMOVE	Removes a specified job from the job queue.	on page 8-11
CHANGE	Alters a specified job. You can alter the job description, the time at which the job will be run, or the interval between executions of the job.	on page 8-11
WHAT	Alters the job description for a specified job.	on page 8-11
NEXT_DATE	Alters the next execution time for a specified job.	on page 8-12
INTERVAL	Alters the interval between executions for a specified job.	on page 8-12
BROKEN	Disables job execution. If a job is marked as broken, Oracle does not attempt to execute it.	on page 8-12
RUN	Forces a specified job to run.	on page 8-13

Submitting a Job to the Job Queue

To submit a new job to the job queue, use the SUBMIT procedure in the DBMS_JOB package:

```
DBMS_JOB.SUBMIT(  job            OUT    BINARY_INTEGER,
                  what            IN    VARCHAR2,
                  next_date       IN    DATE DEFAULT SYSDATE,
```

```
interval        IN    VARCHAR2 DEFAULT 'null',
no_parse        IN    BOOLEAN DEFAULT FALSE)
```

The SUBMIT procedure returns the number of the job you submitted. Table 8–3 describes the procedure's parameters.

Table 8–3 Parameters for DBMS_JOB.SUBMIT

Parameter	Description
job	This is the identifier assigned to the job you created. You must use the job number whenever you want to alter or remove the job. For more information about job numbers, see "Job Numbers" on page 8-7.
what	This is the PL/SQL code you want to have executed. For more information about defining a job, see "Job Definitions" on page 8-7.
next_date	This is the next date when the job will be run. The default value is SYSDATE.
interval	This is the date function that calculates the next time to execute the job. The default value is NULL. INTERVAL must evaluate to a future point in time or NULL. For more information on how to specify an execution interval, see "Job Execution Interval" on page 8-8.
no_parse	This is a flag. The default value is FALSE. If NO_PARSE is set to FALSE (the default), Oracle parses the procedure associated with the job. If NO_PARSE is set to TRUE, Oracle parses the procedure associated with the job the first time that the job is executed. If, for example, you want to submit a job before you have created the tables associated with the job, set NO_PARSE to TRUE.

As an example, let's submit a new job to the job queue. The job calls the procedure DBMS_DDL.ANALYZE_OBJECT to generate optimizer statistics for the table DQUON.ACCOUNTS. The statistics are based on a sample of half the rows of the ACCOUNTS table. The job is run every 24 hours:

```
VARIABLE jobno number;
begin
        2>          DBMS_JOB.SUBMIT(:jobno,
        3>                 'dbms_ddl.analyze_object(''TABLE'',
        4>                  ''DQUON'', ''ACCOUNTS'',
        5>                  ''ESTIMATE'', NULL, 50);'
        6>                 SYSDATE, 'SYSDATE + 1');
        7>           commit;
```

```
          8>  end;
          9>  /
Statement processed.
print jobno
JOBNO
----------
14144
```

Job Environment

When you submit a job to the job queue or alter a job's definition, Oracle records the following environment characteristics:

- the current user
- the user submitting or altering a job
- the current schema
- MAC privileges (if appropriate)

Oracle also records the following NLS parameters:

- NLS_LANGUAGE
- NLS_TERRITORY
- NLS_CURRENCY
- NLS_ISO_CURRENCY
- NLS_NUMERIC_CHARACTERS
- NLS_DATE_FORMAT
- NLS_DATE_LANGUAGE
- NLS_SORT

Oracle restores these environment characteristics every time a job is executed. NLS_LANGUAGE and NLS_TERRITORY parameters are defaults for unspecified NLS parameters.

You can change a job's environment by using the DBMS_SQL package and the ALTER SESSION command.

Jobs and Import/Export

Jobs can be exported and imported. Thus, if you define a job in one database, you can transfer it to another database. When exporting and importing jobs, the job's number, environment, and definition remain unchanged.

> **Note:** If the job number of a job you want to import matches the number of a job already existing in the database, you will not be allowed to import that job. Submit the job as a new job in the database.

Job Owners

When you submit a job to the job queue, Oracle identifies you as the owner of the job. Only a job's owner can alter the job, force the job to run, or remove the job from the queue.

Job Numbers

A queued job is identified by its job number. When you submit a job, its job number is automatically generated from the sequence SYS.JOBSEQ.

Once a job is assigned a job number, that number does not change. Even if the job is exported and imported, its job number remains the same.

Job Definitions

The *job definition* is the PL/SQL code specified in the WHAT parameter of the SUBMIT procedure.

Normally the job definition is a single call to a procedure. The procedure call can have any number of parameters.

> **Note:** In the job definition, use two single quotation marks around strings. Always include a semicolon at the end of the job definition.

There are special parameter values that Oracle recognizes in a job definition. Table 8–4 lists these parameters.

Table 8–4 Special Parameter Values for Job Definitions

Parameter	Mode	Description
job	IN	The number of the current job.
next_date	IN / OUT	The date of the next execution of the job. The default value is SYSDATE.
broken	IN / OUT	Status of job, broken or not broken. The IN value is FALSE.

The following are examples of valid job definitions:

```
'myproc(''10-JAN-82'', next_date, broken);'
'scott.emppackage.give_raise(''JFEE'', 3000.00);'
'dbms_job.remove(job);'
```

Job Execution Interval

The INTERVAL date function is evaluated immediately before a job is executed. If the job completes successfully, the date calculated from INTERVAL becomes the new NEXT_DATE. If the INTERVAL date function evaluates to NULL and the job completes successfully, the job is deleted from the queue.

If a job should be executed periodically at a set interval, use a date expression similar to 'SYSDATE + 7' in the INTERVAL parameter. For example, if you set the execution interval to 'SYSDATE + 7' on Monday, but for some reason (such as a network failure) the job is not executed until Thursday, 'SYSDATE + 7' then executes every Thursday, not Monday.

If you always want to automatically execute a job at a specific time, regardless of the last execution (for example, every Monday), the INTERVAL and NEXT_DATE parameters should specify a date expression similar to 'NEXT_DAY(TRUNC(SYSDATE), "MONDAY")'.

Table 8–5 lists some common date expressions used for job execution intervals.

Table 8–5 Common Job Execution Intervals

Date Expression	Evaluation
'SYSDATE + 7'	exactly seven days from the last execution
'SYSDATE + 1/48'	every half hour
'NEXT_DAY(TRUNC(SYSDATE), ''MONDAY'') + 15/24'	every Monday at 3PM
'NEXT_DAY(ADD_MONTHS (TRUNC(SYSDATE, ''Q''), 3), ''THURSDAY'')'	first Thursday of each quarter

Note: When specifying NEXT_DATE or INTERVAL, remember that date literals and strings must be enclosed in single quotation marks. Also, the value of INTERVAL must be enclosed in single quotation marks.

Database Links and Jobs

If you submit a job that uses a database link, the link must include a username and password. Anonymous database links will not succeed.

See Also: For more information on the DBMS_SQL package, see the *Oracle8i Supplied Packages Reference.*

How Jobs Execute

SNP background processes execute jobs. To execute a job, the process creates a session to run the job.

When an SNP process runs a job, the job is run in the same environment in which it was submitted and with the owner's default privileges.

When you force a job to run using the procedure DBMS_JOB.RUN, the job is run by your user process. When your user process runs a job, it is run with your default privileges only. Privileges granted to you through roles are unavailable.

Job Queue Locks

Oracle uses job queue locks to ensure that a job is executed one session at a time. When a job is being run, its session acquires a job queue (JQ) lock for that job.

Interpreting Information about JQ Locks You can use the Enterprise Manager Lock Monitor or the locking views in the data dictionary to examine information about locks currently held by sessions.

The following query lists the session identifier, lock type, and lock identifiers for all sessions holding JQ locks:

```
SELECT sid, type, id1, id2
   FROM v$lock
   WHERE type = 'JQ';

SID         TY    ID1        ID2
----------   --   ---------- ----------
        12   JQ   0          14144
1 row selected.
```

In the query above, the identifier for the session holding the lock is 12. The ID1 lock identifier is always 0 for JQ locks. The ID2 lock identifier is the job number of the job the session is running.

Job Execution Errors

When a job fails, information about the failure is recorded in a trace file and the alert log. Oracle writes message number ORA-12012 and includes the job number of the failed job.

The following can prevent the successful execution of queued jobs:

- not having any SNP background processes to run the job
- a network or instance failure
- an exception when executing the job

Job Failure and Execution Times If a job returns an error while Oracle is attempting to execute it, Oracle tries to execute it again. The first attempt is made after one minute, the second attempt after two minutes, the third after four minutes, and so on, with the interval doubling between each attempt. When the retry interval exceeds the execution interval, Oracle continues to retry the job at the normal execution interval. However, if the job fails 16 times, Oracle automatically marks the job as broken and no longer tries to execute it.

Thus, if you can correct the problem that is preventing a job from running before the job has failed sixteen times, Oracle will eventually run that job again.

See Also: For more information about the locking views, see the *Oracle8i Reference*.

For more information about locking, see *Oracle8i Concepts*.

Removing a Job from the Job Queue

To remove a job from the job queue, use the REMOVE procedure in the DBMS_JOB package:

```
DBMS_JOB.REMOVE(job IN BINARY_INTEGER)
```

The following statement removes job number 14144 from the job queue:

```
DBMS_JOB.REMOVE(14144);
```

Restrictions

You can remove currently executing jobs from the job queue. However, the job will not be interrupted, and the current execution will be completed.

You can remove only jobs you own. If you try to remove a job that you do not own, you receive a message that states the job is not in the job queue.

Altering a Job

To alter a job that has been submitted to the job queue, use the procedures CHANGE, WHAT, NEXT_DATE, or INTERVAL in the DBMS_JOB package.

In this example, the job identified as 14144 is now executed every three days:

```
DBMS_JOB.CHANGE(14144, null, null, 'SYSDATE + 3');
```

Restrictions

You can alter only jobs that you own. If you try to alter a job that you do not own, you receive a message that states the job is not in the job queue.

Syntax for CHANGE

You can alter any of the user-definable parameters associated with a job by calling the DBMS_JOB.CHANGE procedure. Table 8–3 describes the procedure's parameters.

```
DBMS_JOB.CHANGE( job                    IN  BINARY_INTEGER,
          what                          IN  VARCHAR2,
          next_date                     IN  DATE,
          interval                      IN  VARCHAR2)
```

If you specify NULL for WHAT, NEXT_DATE, or INTERVAL when you call the procedure CHANGE, the current value remains unchanged.

> **Note:** When you change a job's definition using the WHAT parameter in the procedure CHANGE, Oracle records your current environment. This becomes the new environment for the job.

Syntax for WHAT

You can alter the definition of a job by calling the DBMS_JOB.WHAT procedure. Table 8–3 describes the procedure's parameters.

```
DBMS_JOB.WHAT( job                  IN  BINARY_INTEGER,
        what                        IN  VARCHAR2)
```

> **Note:** When you execute procedure WHAT, Oracle records your current environment. This becomes the new environment for the job.

Syntax for NEXT_DATE

You can alter the next date that Oracle executes a job by calling the DBMS_JOB.NEXT_DATE procedure. Table 8–3 describes the procedure's parameters.

```
DBMS_JOB.NEXT_DATE( job       IN  BINARY_INTEGER,
next_date                     IN  DATE)
```

Syntax for INTERVAL

You can alter the execution interval of a job by calling the DBMS_JOB.INTERVAL procedure. Table 8–3 describes the procedure's parameters.

```
DBMS_JOB.INTERVAL( job              IN BINARY_INTEGER,
        interval                    IN VARCHAR2)
```

Broken Jobs

A job is labeled as either broken or not broken. Oracle does not attempt to run broken jobs. However, you can force a broken job to run by calling the procedure DBMS_JOB.RUN.

When you submit a job it is considered not broken.

There are two ways a job can break:

- Oracle has failed to successfully execute the job after 16 attempts.
- You have marked the job as broken, using the procedure DBMS_JOB.BROKEN.

To mark a job as broken or not broken, use the procedure BROKEN in the DBMS_JOB package. Table 8–4 describes the procedure's parameters:

```
DBMS_JOB.BROKEN( job              IN BINARY_INTEGER,
        broken                    IN BOOLEAN,
        next_date                 IN DATE DEFAULT SYSDATE)
```

The following example marks job 14144 as not broken and sets its next execution date to the following Monday:

```
DBMS_JOB.BROKEN(14144, FALSE, NEXT_DAY(SYSDATE, 'MONDAY'));
```

Once a job has been marked as broken, Oracle will not attempt to execute the job until you either mark the job as not broken, or force the job to be executed by calling the procedure DBMS_JOB.RUN.

Restrictions

You can mark as broken only jobs that you own. If you try to mark a job you do not own, you receive a message stating that the job is not in the job queue.

Running Broken Jobs

If a problem has caused a job to fail 16 times, Oracle marks the job as broken. Once you have fixed this problem, you can run the job by either:

- forcing the job to run by calling DBMS_JOB.RUN
- marking the job as not broken by calling DBMS_JOB.BROKEN and waiting for Oracle to execute the job

If you force the job to run by calling the procedure DBMS_JOB.RUN, Oracle runs the job immediately. If the job succeeds, then Oracle labels the job as not broken and resets its count of the number of failed executions for the job.

Once you reset a job's broken flag (by calling either RUN or BROKEN), job execution resumes according to the scheduled execution intervals set for the job.

Forcing a Job to Execute

There may be times when you would like to manually execute a job. For example, if you have fixed a broken job, you may want to test the job immediately by forcing it to execute.

To force a job to be executed immediately, use the procedure RUN in the DBMS_JOB package. Oracle attempts to run the job, even if the job is marked as broken:

```
DBMS_JOB.RUN( job  IN BINARY_INTEGER)
```

When you run a job using DBMS_JOB.RUN, Oracle recomputes the next execution date. For example, if you create a job on a Monday with a NEXT_DATE value of 'SYSDATE' and an INTERVAL value of 'SYSDATE + 7', the job is run every 7 days starting on Monday. However, if you execute RUN on Wednesday, the next execution date will be the next Wednesday.

Note: When you force a job to run, the job is executed in your current session. Running the job reinitializes your session's packages.

Restrictions

You can only run jobs that you own. If you try to run a job that you do not own, you receive a message that states the job is not in the job queue.

The following statement runs job 14144 in your session and recomputes the next execution date:

```
DBMS_JOB.RUN(14144);
```

The procedure RUN contains an implicit commit. Once you execute a job using RUN, you cannot roll back.

Terminating a Job

You can terminate a running job by marking the job as broken, identifying the session running the job, and disconnecting that session. You should mark the job as broken, so that Oracle does not attempt to run the job again.

After you have identified the session running the job (via V$SESSION), you can disconnect the session using the SQL statement ALTER SYSTEM.

See Also: For examples of viewing information about jobs and sessions, see the following section, "Viewing Job Queue Information".

For more information on V$SESSION, see the *Oracle8i Reference*.

Viewing Job Queue Information

You can view information about jobs in the job queue via the data dictionary views in Table 8–6:

Table 8–6 Views for Job Queue Information

View	Description
DBA_JOBS	Lists all the jobs in the database.
USER_JOBS	Lists all jobs owned by the user.
DBA_JOBS_RUNNING	Lists all jobs in the database that are currently running. This view joins V$LOCK and JOB$.

For example, you can display information about a job's status and failed executions. The following sample query creates a listing of the job number, next execution time, failures, and broken status for each job you have submitted:

```
SELECT job, next_date, next_sec, failures, broken
   FROM user_jobs;

JOB        NEXT_DATE NEXT_SEC FAILURES    B
---------- --------- -------- ----------  -
  9125    01-NOV-94 00:00:00      4       N
 14144    24-OCT-94 16:35:35      0       N
 41762    01-JAN-00 00:00:00     16       Y
3 rows selected.
```

You can also display information about jobs currently running. The following sample query lists the session identifier, job number, user who submitted the job, and the start times for all currently running jobs:

```
SELECT sid, r.job, log_user, r.this_date, r.this_sec
  FROM dba_jobs_running r, dba_jobs j
  WHERE r.job = j.job;


SID        JOB        LOG_USER            THIS_DATE  THIS_SEC
---------- ---------- -------------------- ---------   --------
 12        14144     JFEE                 24-OCT-94  17:21:24
 25         8536     SCOTT                24-OCT-94  16:45:12
2 rows selected.
```

See Also: For more information on data dictionary views, see the *Oracle8i Reference*.

Part III

Database Storage

9

Managing Tablespaces

This chapter describes the various aspects of tablespace management, and includes the following topics:

- Guidelines for Managing Tablespaces
- Creating Tablespaces
- Managing Tablespace Allocation
- Altering Tablespace Availability
- Making a Tablespace Read-Only
- Dropping Tablespaces
- Using the DBMS_SPACE_ADMIN Package
- Transporting Tablespaces Between Databases
- Viewing Information About Tablespaces

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
- Specifying Tablespace Storage Parameters
- Assigning Tablespace Quotas to Users

Using Multiple Tablespaces

Using multiple tablespaces allows you more flexibility in performing database operations. For example, when a database has multiple tablespaces, you can perform the following tasks:

- Separate user data from data dictionary data.
- Separate one application's data from another's.
- Store different tablespaces' datafiles on separate disk drives to reduce I/O contention.
- Separate rollback segment data from user data, preventing a single disk failure from causing permanent loss of data.
- Take individual tablespaces offline while others remain online.
- Reserve 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 simultaneously open; these limits can affect the number of tablespaces that can be simultaneously online. To avoid exceeding your operating system's 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 the autoextend option set on, rather than many small datafiles.

Review your data in light of these advantages and decide how many tablespaces you will need for your database design.

Specifying Tablespace Storage Parameters

When you create a new 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. However, if you do not specify storage parameters when creating an object, the object's 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.

> **Note:** If you do not specify the default storage parameters for a new tablespace, the default storage parameters of Oracle become the tablespace's default storage parameters.

See Also: For information about estimating the sizes of objects, see Chapters 11 through 17.

Assigning Tablespace Quotas to Users

Grant to users who will be creating tables, clusters, snapshots, 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's segment. The security administrator is responsible for granting the required privileges to create objects to database users and for assigning tablespace quotas, as necessary, to database users.

See Also: To learn more about assigning tablespace quotas to database users, see "Assigning Tablespace Quotas" on page 23-13.

Creating Tablespaces

The steps for creating tablespaces vary by operating system. On most operating systems you indicate the size and fully specified filenames when creating a new tablespace or altering a tablespace by adding datafiles. In each situation Oracle automatically allocates and formats the datafiles as specified. However, on some operating systems, you must create the datafiles before installation.

The first tablespace in any database is always the SYSTEM tablespace. Therefore, the first datafiles of any database are automatically allocated for the SYSTEM tablespace during database creation.

You might create a new tablespace for any of the following reasons:

- You want to allocate more disk storage space for the associated database, thereby enlarging the database.
- You need to create a logical storage structure in which to store a specific type of data separate from other database data.

To increase the total size of the database you can alternatively add a datafile to an existing tablespace, rather than adding a new tablespace.

> **Note:** No data can be inserted into any tablespace until the current instance has acquired at least two rollback segments (including the SYSTEM rollback segment).

To create a new tablespace, use the SQL statement CREATE TABLESPACE. You must have the CREATE TABLESPACE system privilege to create a tablespace.

As an example, let's create the tablespace RB_SEGS (to hold rollback segments for the database), with the following characteristics:

- The data of the new tablespace is contained in a single datafile, 50M in size.
- The default storage parameters for any segments created in this tablespace are explicitly set.
- After the tablespace is created, it is left offline.

The following statement creates the tablespace RB_SEGS:

```
CREATE TABLESPACE rb_segs
    DATAFILE 'datafilers_1' SIZE 50M
    DEFAULT STORAGE (
      INITIAL 50K
      NEXT 50K
      MINEXTENTS 2
      MAXEXTENTS 50
      PCTINCREASE 0)
    OFFLINE;
```

If you do not fully specify filenames when creating tablespaces, the corresponding datafiles are created in the `ORACLE_HOME/dbs` directory.

See Also: See your operating system-specific Oracle documentation for information about initially creating a tablespace.

For more information about adding a datafile, see "Creating and Adding Datafiles to a Tablespace" on page 10-5.

For more information about the CREATE TABLESPACE statement, see the *Oracle8i SQL Reference*.

Creating Locally Managed Tablespaces

Typically, tablespaces are "dictionary mapped," which means that such tablespaces rely on SQL dictionary tables to track space utilization. *Locally managed tablespaces*, on the other hand, use bit maps (instead of SQL dictionary tables) to track used and free space.

Creating and using locally managed tablespaces offers you the following benefits:

- Improves concurrence of space operations

  You allocate and deallocate space by changing the bit values (0 to 1 for allocation, 1 to 0 for deallocation).

- Eliminates recursion during space management operations
- Supports temporary tablespace management in standby databases
- Reduces user reliance on the data dictionary

  Necessary information is stored in segment headers and bit map blocks.

The following statement creates a locally managed tablespace named TBS_1; every extent is 128K, and each bit in the bit map describes 64 blocks:

```
CREATE TABLESPACE tbs_1 DATAFILE 'file_1.f'
   BITMAP ALLOCATION UNIFORM SIZE 128K;
```

See Also: For detailed syntax on creating locally managed tablespaces, see the *Oracle8i SQL Reference*.

Creating a Database with a Locally Managed SYSTEM Tablespace

You can create a database with a locally managed SYSTEM tablespace. However, rollback segments for this database must also be created in uniform-managed locally managed tablespaces. A locally managed SYSTEM tablespace is always system-managed. Also, you cannot later revert to a version of Oracle earlier than 8.1.

See Also: For more information about creating a database with a locally managed SYSTEM tablespace, see the *Oracle8i SQL Reference.*

Creating a Temporary Tablespace

If you wish to improve the concurrence of multiple sort operations, reduce their overhead, or avoid Oracle space management operations altogether, you can create *temporary tablespaces*.

Within a temporary tablespace, all sort operations for a given instance and tablespace share a single *sort segment*. Sort segments exist in every instance that performs sort operations within a given tablespace. You cannot store permanent objects in a temporary tablespace. You can view the allocation and deallocation of space in a temporary tablespace sort segment via the V$SORT_SEGMENT table.

To identify a tablespace as temporary during tablespace creation, issue the following statement:

```
CREATE TABLESPACE tablespace TEMPORARY;
```

To identify a tablespace as temporary in an existing tablespace, issue the following statement:

```
ALTER TABLESPACE tablespace TEMPORARY;
```

Note: You can take temporary tablespaces offline. Returning temporary tablespaces online does not affect their temporary status.

See Also: For more information about the CREATE TABLESPACE and ALTER TABLESPACE statements, see the *Oracle8i SQL Reference.*

For more information about V$SORT_SEGMENT, see the *Oracle8i Reference.*

For more information about Oracle space management, see *Oracle8i Concepts.*

Temporary Datafiles

Temporary datafiles differ from permanent datafiles in that they do not appear in the DBA_DATA_FILES view. Instead, they appear in the DBA_TEMP_FILES view, which is similar to DBA_DATA_FILES view except that it contains information about temporary datafiles. In SQL, files belonging to temporary tablespaces are also identified as TEMPFILES, rather than DATAFILES.

See Also: For more information about temporary datafiles and DBA_TEMP_FILES, see the *Oracle8i Reference.*

Creating a Locally Managed Temporary Tablespace

If you wish to allocate space that can contain schema objects for the duration of a session in the database, you can create a locally managed temporary tablespace.

You must have the CREATE TABLESPACE system privilege to create a locally managed temporary tablespace.

The following statement creates a temporary tablespace in which each extent is 16M. The default database block size is 2M; each bit in the map represents one extent, thus each bit maps 8,000 blocks.

```
CREATE TEMPORARY TABLESPACE tbs_1 TEMPFILE 'file_1.f'
   BITMAP ALLOCATION UNIFORM SIZE 16M;
```

See Also: For more information about creating a locally managed temporary tablespace, see the *Oracle8i SQL Reference.*

Altering a Locally Managed Temporary Tablespace

You can alter or add a datafile (or temporary file) to a locally managed temporary tablespace.

The following statement adds files to a locally managed temporary tablespace:

```
ALTER TABLESPACE tbs_1
   ADD TEMPFILE 'file_1.f';
```

The following statements take offline and bring online temporary files:

```
ALTER DATABASE TEMPFILE 'temp_file_1.f' OFFLINE;
ALTER DATABASE TEMPFILE 'temp_file_1.f' ONLINE;
```

The following statement resizes temporary file TEMP_FILE_1.F to 12K:

```
ALTER DATABASE TEMPFILE 'temp_file_1.f' RESIZE 12K;
```

The following statement drops a temporary file:

```
ALTER DATABASE TEMPFILE 'temp_file_1.f' DROP;
```

See Also: For details and restrictions about statements used to alter locally managed temporary tablespaces, see the *Oracle8i SQL Reference.*

Managing Tablespace Allocation

This section describes aspects of managing tablespace allocation, and includes the following topics:

- Altering Storage Settings for Tablespaces
- Coalescing Free Space

Altering Storage Settings for Tablespaces

You can change the default storage parameters of a tablespace to change the default specifications for *future* objects created in the tablespace. To change the default storage parameters for objects subsequently created in the tablespace, use the SQL statement ALTER TABLESPACE. Also, to alter the default storage parameters of a tablespace, you must have the ALTER TABLESPACE system privilege.

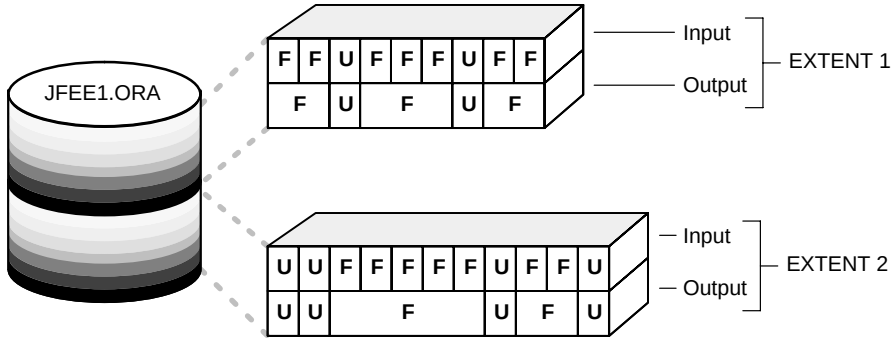
```
ALTER TABLESPACE users
   DEFAULT STORAGE (
     INITIAL 50K
     NEXT 50K
     MINEXTENTS 2
     MAXEXTENTS 20
     PCTINCREASE 50);
```

New values for the default storage parameters of a tablespace affect only future extents allocated for the segments within the tablespace.

Coalescing Free Space

Space for tablespace segments is managed using extents, which are made up of a specific number of contiguous data blocks. The free extent closest in size to the required extent is used when allocating new extents to a tablespace segment. Thus, a larger free extent can be fragmented, or smaller contiguous free extents can be coalesced into one larger free extent (see Figure 9–1). However, continuous allocation and deallocation of free space fragments your tablespace and makes allocation of larger extents more difficult. By default, SMON (system monitor) processes incrementally coalesce the free extents of tablespaces in the background. If desired, you can disable SMON coalescing.

Figure 9–1 Coalescing Free Space



F = free data block
U = used data block

If you find that fragmentation of space is high (contiguous space on your disk appears as non-contiguous), you can coalesce your free space in a single space transaction. After every eight coalesces the space transaction commits and other transactions can allocate or deallocate space. You must have ALTER TABLESPACE privileges to coalesce tablespaces. You can coalesce all available free space extents in a tablespace into larger contiguous extents on a per tablespace basis by using the following command:

```
ALTER TABLESPACE tablespace COALESCE;
```

You can also use this command to supplement SMON and extent allocation coalescing, thereby improving space allocation performance in severely fragmented tablespaces. Issuing this command does not effect the performance of other users accessing the same tablespace. Like other options of the ALTER TABLESPACE statement, the COALESCE option is exclusive; when specified, it should be the only option.

Viewing Information about Tablespaces

To display statistics about coalesceable extents for tablespaces, you can view the DBA_FREE_SPACE_COALESCED view. You can query this view to determine if you need to coalesce space in a particular tablespace.

See Also: For information about the contents of DBA_FREE_SPACE_COALESCED, see the *Oracle8i Reference*.

Altering Tablespace Availability

You can bring an offline tablespace online to make the schema objects within the tablespace available to database users. Alternatively, you can take an online tablespace offline while the database is open, so that this portion of the database is temporarily unavailable for general use but the rest is open and available. This section includes the following topics:

- Bringing Tablespaces Online
- Taking Tablespaces Offline

Bringing Tablespaces Online

You can bring any tablespace in an Oracle database online whenever the database is open. The only exception is that the SYSTEM tablespace must always be online because the data dictionary must always be available to Oracle. A tablespace is normally online so that the data contained within it is available to database users.

To bring an offline tablespace online while the database is open, use the SQL statement ALTER TABLESPACE. You must have the MANAGE TABLESPACE system privilege to bring a tablespace online.

> **Note:** If a tablespace to be brought online was not taken offline "cleanly" (that is, using the NORMAL option of the ALTER TABLESPACE OFFLINE statement), you must first perform media recovery on the tablespace before bringing it online. Otherwise, Oracle returns an error and the tablespace remains offline.

The following statement brings the USERS tablespace online:

```
ALTER TABLESPACE users ONLINE;
```

Taking Tablespaces Offline

You may wish 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.

To take an online tablespace offline while the database is open, use the SQL command ALTER TABLESPACE. You must have the MANAGE TABLESPACE system privilege to take a tablespace offline.

You can specify any of the following priorities when taking a tablespace offline:

normal offline	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. With normal offline priority, Oracle takes a checkpoint for all datafiles of the tablespace as it takes them offline.
temporary offline	A tablespace can be taken offline temporarily, even if there are error conditions for one or more files of the tablespace. With temporary offline priority, Oracle 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 option, 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 will require recovery before you can bring it back online.
immediate offline	A tablespace can be taken offline immediately, without Oracle's taking a checkpoint on any of the datafiles. With immediate offline priority, 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.

WARNING: If you must take a tablespace offline, use the normal option (the default) if possible; this guarantees that the tablespace will not require recovery to come back online, even if you reset the redo log sequence (using an ALTER DATABASE OPEN RESETLOGS statement after incomplete media recovery) before bringing the tablespace back online.

Take a tablespace offline temporarily only when you cannot take it offline normally; in this case, only the files taken offline because of errors need to be recovered before the tablespace can be brought online. Take a tablespace offline immediately only after trying both the normal and temporary options.

The following example takes the USERS tablespace offline normally:

```
ALTER TABLESPACE users OFFLINE NORMAL;
```

See Also: Before taking an online tablespace offline, verify that the tablespace contains no active rollback segments. For more information see "Taking Rollback Segments Offline" on page 21-12.

Making a Tablespace Read-Only

This section describes issues related to making tablespaces read-only, and includes the following topics:

- Prerequisites
- Making a Read-Only Tablespace Writeable
- Creating a Read-Only Tablespace on a WORM Device

Making a tablespace read-only prevents further write operations on the datafiles in the tablespace. After making the tablespace read-only, you should back it up.

Use the SQL statement ALTER TABLESPACE to change a tablespace to read-only. You must have the ALTER TABLESPACE system privilege to make a tablespace read-only. The following statement makes the FLIGHTS tablespace read-only:

```
ALTER TABLESPACE flights READ ONLY
```

After a tablespace is read-only, you can copy its files to read-only media. You must then rename the datafiles in the control file to point to the new location by using the SQL statement ALTER DATABASE RENAME.

A read-only tablespace is neither online nor offline. Issuing the ALTER TABLESPACE statement with the ONLINE or OFFLINE option does not change the read-only state of the tablespace; rather, it causes all of the datafiles in the tablespace to be brought online or offline.

The ALTER TABLESPACE...READ ONLY statement waits until active transactions are complete before performing the read-only operation. Thus, you do not have to wait for transactions to complete before making a tablspace read-only.

Prerequisites

Before you can make a tablespace read-only, the following conditions must be met. It may be easiest to meet these restrictions by performing this function in restricted mode, so that only users with the RESTRICTED SESSION system privilege can be logged on.

- 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 must not contain any active rollback segments.

  For this reason, the SYSTEM tablespace can never be made read-only, since it contains the SYSTEM rollback segment. Additionally, because the rollback segments of a read-only tablespace are not accessible, it is recommended that you drop the rollback segments before you make a tablespace read-only.

- The tablespace must not currently be involved in an online backup, since the end of a backup updates the header file of all datafiles in the tablespace.

- The COMPATIBLE initialization parameter must be set to 7.1.0 or greater.

For better performance while accessing data in a read-only tablespace, you might want to 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 will ensure that the data blocks in the tablespace can be subsequently accessed most efficiently. This eliminates the need for Oracle to check the status of the transactions that most recently modified the blocks.

WARNING: You cannot rename or resize datafiles belonging to a read-only tablespace.

See Also: For more information about read-only tablespaces, see *Oracle8i Concepts.*

Making a Read-Only Tablespace Writeable

Whenever you create a tablespace, it is both readable and writeable. To change a read-only tablespace back to a read-write tablespace, use the SQL command ALTER TABLESPACE. You must have the ALTER TABLESPACE system privilege to change a read-only tablespace to a read-write tablespace. The following command makes the FLIGHTS tablespace writeable:

```
ALTER TABLESPACE flights READ WRITE;
```

Making a read-only tablespace writeable updates the control file for the datafiles, so that you can use the read-only version of the datafiles as a starting point for recovery.

Prerequisites

To issue this command, all of the datafiles in the tablespace must be online. Use the DATAFILE ONLINE option of the ALTER DATABASE command to bring a datafile online. The V$DATAFILE view lists the current status of a datafile.

Creating a Read-Only Tablespace on a WORM Device

To create a read-only tablespace on a WORM (Write Once Read Many) device when you have read-only files that do not require updating:

1. Create a writeable tablespace on another device. Create the objects that belong in the tablespace and insert your data.
2. Issue the ALTER TABLESPACE command with the READ ONLY option to change the tablespace to 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. Renaming the datafiles changes their names in the control file.
6. Bring the tablespace online.

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. Any tablespace in an Oracle database, except the SYSTEM tablespace, can be dropped. You must have the DROP TABLESPACE system privilege to drop a tablespace.

> **WARNING: Once a tablespace has been dropped, the tablespace's data 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, only the file pointers in the control files of the associated database are dropped. The datafiles that constituted the dropped tablespace continue to exist. To free previously used disk space, delete the datafiles of the dropped tablespace using the appropriate commands of your operating system after completing this procedure.

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 an active rollback segment, you cannot drop the tablespace. For simplicity, take the tablespace offline before dropping it.

After a tablespace is dropped, the tablespace's entry remains in the data dictionary (see the DBA_TABLESPACES view), but the tablespace's status is changed to INVALID.

To drop a tablespace, use the SQL command DROP TABLESPACE. 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 check the Including Contained Objects checkbox. If the tablespace contains any tables with primary or unique keys referenced by foreign keys of tables in other tablespaces and you want to cascade the drop of the FOREIGN KEY constraints of the child tables, select the Cascade Drop of Integrity Constraints checkbox to drop the tablespace.

Use the CASCADE CONSTRAINTS option of the DROP TABLESPACE statement to cascade the drop of the FOREIGN KEY constraints in the child tables.

See Also: For more information about taking tablespaces offline, see "Taking Tablespaces Offline" on page 9-10.

For more information about the DROP TABLESPACE statement, see the *Oracle8i SQL Reference*.

Using the DBMS_SPACE_ADMIN Package

The DBMS_SPACE_ADMIN package provides administrators with defect diagnosis and repair functionality. The following scenarios describe typical situations in which you can use the DBMS_SPACE_ADMIN package to diagnose and resolve problems.

The DBMS_SPACE_ADMIN package contains the following procedures:

- SEGMENT_VERIFY
- SEGMENT_CORRUPT
- SEGMENT_DROP_CORRUPT
- SEGMENT_DUMP
- TABLESPACE_VERIFY
- TABLESPACE_REBUILD_BITMAPS
- TABLESPACE_FIX_BITMAPS
- TABLESPACE_MIGRATE_TO_BITMAP
- TABELSPACE_MIGRATE_FROM_BITMAP

See Also: For details about these procedures, see the *Oracle8i Supplied Packages Reference*.

Scenario 1

The TABLESPACE_VERIFY procedure discovers that a segment has allocated blocks that are marked "free" in the bit map, but no overlap between segments was reported.

In this scenario, perform the following tasks:

- Call the SEGMENT_EXTENT_MAP_DUMP procedure to dump the ranges that the administrator allocated to the segment.

- For each range, call the TABLESPACE_FIX_BITMAPS procedure with the TABLESPACE_MAKE_USED option to mark the space as used.

Scenario 2

You cannot drop a segment because the bit map has segment blocks marked "free." The system has automatically marked it corrupt.

In this scenario, perform the following tasks:

- Call the SEGMENT_VERIFY procedure with the SEGMENT_CHECK_ALL option. If no overlaps are reported, perform the following:
  - Call the SEGMENT_EXTENT_MAP_DUMP procedure to dump the ranges that the administrator allocated to the segment.
  - For each range, call the TABLESPACE_FIX_BITMAPS procedure with the TABLESPACE_MAKE_FREE option to mark the space as "free."
  - Call the SEGMENT_DROP_CORRUPT procedure to drop the SEG$ entry.

Scenario 3

The TABLESPACE_VERIFY procedure has reported some overlapping. Some of the real data must be sacrificed based on previous internal errors.

After choosing the object to be sacrificed, say table T1, perform the following tasks:

- Make a list of all objects that T1 overlaps.
- Drop table T1. If necessary, follow up by calling the SEGMENT_DROP_CORRUPT procedure.
- Call the SEGMENT_VERIFY procedure on all objects that T1 overlapped. If necessary, call the TABLESPACE_FIX_BITMAPS procedure to mark appropriate bit maps as used.
- Rerun the TABLESPACE_VERIFY procedure to verify the problem is resolved.

Scenario 4

A set of bitmap blocks has media corruption.

In this scenario, perform the following tasks:

- Call the TABLESPACE_REBUILD_MAPS procedure, either on all bitmap blocks, or on a single block if only one is corrupt.

- Call the TABLESPACE_VERIFY procedure to verify that the bit maps are consistent.

See Also: For more information about the DBMS_SPACE_ADMIN package, see the *Oracle8i Supplied Packages Reference.*

Transporting Tablespaces Between Databases

This section describes how to transport tablespaces between databases, and includes the following topics:

Introduction to Transportable Tablespaces

Note: You must have the Oracle8i Enterprise Edition of Oracle to generate a transportable tablespace set. However, you can use any edition of Oracle (Enterprise, Work group, or Personal Oracle8i) to plug a transportable tablespace set into an Oracle database.

You can use *transportable tablespaces* to move a subset of an Oracle database and "plug" it in to another Oracle database, essentially moving tablespaces between the databases. Transporting tablespaces is particularly useful for:

- Feeding data from OLTP systems to data warehouse staging systems
- Updating data warehouses and data marts from staging systems
- Loading data marts from central data warehouses
- Archiving OLTP and data warehouse systems efficiently
- Data publishing to internal and external customers

Moving data via transportable tablespaces can be much faster than performing either an import/export or unload/load of the same data, because transporting a tablespace only requires the copying of datafiles and integrating the tablespace structural information. You can also use transportable tablespaces to move index data, thereby avoiding the index rebuilds you would have to perform when importing or loading table data.

To move or copy a set of tablespaces you must perform the following tasks:

- Step 1: Pick a Self-contained Set of Tablespaces
- Step 2: Generate a Transportable Tablespace Set

  A *transportable set* consists of datafiles for the set of tablespaces being transported and a file containing structural information for the set of tablespaces.
- Step 3: 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 O/S copying utility, ftp, or publishing on CDs)
- Step 4: Plug In the Tablespace Set

  Invoke Import to plug the set of tablespaces into the target database.

See Also: For more details about transportable tablespaces and their use in data marts and data warehousing, see *Oracle8i Concepts.*

For information about using transportable tablespaces to perform media recovery, see the *Oracle8i Backup and Recovery Guide.*

For information about transportable tablespace compatibility issues (between different Oracle releases), see *Oracle8i Migration.*

Current Limitations

Be aware of the following limitations as you plan for and use transportable tablespaces:

- The source and target database must be on the same hardware platform. For example, you can transport tablespaces between Sun Solaris Oracle databases, or you can transport tablespaces between NT Oracle databases. However, you cannot transport a tablespace from a SUN Solaris Oracle database to an NT Oracle database.
- The source and target database must have the same database block size.
- The source and target database must use the same character set.
- You cannot transport a tablespace to a target database in which a tablespace with the same name already exists.
- Currently, transportable tablespaces do not support:
  - snapshot / replication
  - function-based indexes
  - Scoped REFs
  - domain indexes (a new type of index provided by extensible indexing)
  - 8.0-compatible advanced queues with multiple recipients

Step 1: Pick a Self-contained Set of Tablespaces

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. For example, if there is an index in the set of tablespaces for a table that is outside of the set of tablespaces, then the set of tablespaces is not self-contained.

The tablespace set you wish to copy must contain either all partitions of a partitioned table, or none of the partitions of a partitioned table. If you wish to transport a subset of a partition table, you must exchange the partitions into tables.

When transporting a set of tablespaces, you can choose to include referential integrity constraints. However, doing so can determine 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. Some examples of self contained tablespace violations follow:

- An index inside the set of tablespaces is for a table outside of the set of tablespaces.
- A partitioned table is partially contained in the set of tablespaces.
- A table inside the set of tablespaces contains a LOB column that points to LOBs outside the set of tablespaces.

To determine whether a set of tablespaces is self-contained, you can invoke a built-in PL/SQL procedure, giving it the list of the tablespace names and indicating that you wish to transport referential integrity constraints. For example, suppose you want to determine whether tablespaces `ts1` and `ts2` are self-contained (with constraints taken into consideration). You can issue the following command:

```
execute dbms_tts.transport_set_check('ts1,ts2', TRUE)
```

Here, `transport_set_check` is a PL/SQL routine in the PL/SQL package DBMS_TTS, with the following prototype:

```
PROCEDURE transport_set_check(ts_list IN varchar2, incl_constraints IN boolean)

  ts_list - list of tablespace names separated by comma
  incl_constraints - TRUE if one would like to take constraints into consideration. FALSE
  otherwise.
```

After invoking this PL/SQL routine, you can see all violations by selecting from the TRANSPORT_SET_VIOLATIONS view. If the set of tablespaces is self-contained, this view will be empty. If the set of tablespaces is not self-contained, this view lists all the violations. For example, suppose there are two violations: a foreign key constraint, `dept_fk`, across the tablespace set boundary, and a partitioned table, `sales`, that is partially contained in the tablespace set. Querying TRANSPORT_SET_VIOLATIONS results in the following:

```
select * from transport_set_violations;
VIOLATIONS
-----------------------------------
Constraint DEPT_FK between table JIM.EMP in tablespace FOO and table JIM.DEPT in
tablespace OTHER
Partitioned table JIM.SALES is partially contained in the transportable set
```

Object references (such as REFs) across the tablespace set are not considered violations. REFs are not checked by the TRANSPORT_SET_CHECK routine. When a tablespace containing dangling REFs is plugged into a database, queries following that dangling REF indicate user error.

See Also: For more information about REFs, see the *Oracle8i Application Developer's Guide - Fundamentals.*

Step 2: Generate a Transportable Tablespace Set

After identifying the self-contained set of tablespaces you want to transport, generate a transportable set by performing the following tasks:

1. Make all tablespaces in the set you are copying read-only. Of course, if the tablespaces are already read-only, you do not have to perform this step.

   ```
   ALTER TABLESPACE sales READ ONLY;
   ```

2. Invoke the Export utility and specify which tablespaces are in the transportable set, as follows:

   ```
   EXP TRANSPORT_TABLESPACE=y TABLESPACES=sales_1,sales_2
       TRIGGERS=y/n CONSTRAINTS=y/n GRANTS=y/n FILE=expdat.dmp
   ```

   Note: Although the Export utility is used, only data dictionary structural information is exported. Hence, this operation is even quicker for a large tablespace.

   When prompted, connect as "sys as sysdba."

   You must always specify TABLESPACES. The FILE parameter specifies the name of the structural information export file to be created.

   If you set TRIGGERS=n, triggers are not exported. If you set TRIGGERS=y, triggers are exported without a validity check. Invalid triggers cause compilation errors during the subsequent import.

   If you set GRANTS=y, all grants on the exported tables are exported too; otherwise, all GRANTS are ignored.

   If you set CONSTRAINTS=y, referential integrity constraints are exported; otherwise, referential integrity constraints are ignored.

   The default setting for all of these options is 'y.'

3. Copy the datafiles to a separate storage space or to the target database.

4. If necessary, put the tablespaces in the copied set back into read-write mode as follows:

   ```
   ALTER TABLESPACE sales_1 READ WRITE;
   ```

If the tablespace sets being transported are not self-contained, export will fail and indicate that the transportable set is not self-contained. You must then return to Step 1 to resolve all violations.

Step 3: Transport the Tablespace Set

Transport both the datafiles and the export file to a place accessible to the target database. You can use any facility for copying flat files (for example, an O/S copying utility, ftp, or publishing on CDs).

Step 4: Plug In the Tablespace Set

To plug in a tablespace set, perform the following tasks:

1. Put the copied tablespace set datafiles in a location where the target database can access them.
2. Plug in the tablespaces and integrate the structural information using the following import statement:

```
IMP TRANSPORT_TABLESPACE=y DATAFILES='/db/sales_jan','/db/sales_feb',...fn
    TABLESPACES=sales_1,sales_2,... TTS_OWNERS=dcranney,jfee
    FROMUSER=dcranney,jfee TOUSER=smith,williams FILE=expdat.dmp
```

When prompted, connect as "sys as sysdba."

Following are two more examples:

```
IMP TRANSPORT_TABLESPACE=y DATAFILES='(/db/staging1.f,/db/staging2.f)'

IMP TRANSPORT_TABLESPACE=y DATAFILES='/db/staging.f' TABLESPACES=jan OWNERS=smith
```

You must specify DATAFILES.

TABLESPACES,TTS_OWNERS, FROMUSER and TOUSER are optional. The FILE parameter specifies the name of the structural information export file.

When you specify TABLESPACES, the supplied tablespace names are compared to those in the export file. Import returns an error if there is any mismatch. Otherwise, tablespace names are extracted from the export file.

TTS_OWNERS lists all users who own data in the tablespace set. When you specify TTS_OWNERS, the user names are compared to those in the export file. Import returns an error if there is any mismatch. Otherwise, owner names are extracted from the export file.

If you do not specify FROMUSER and TOUSER, all database objects (such as tables and indexes) will be created under the same user as in the source database. Those users must already exist in the target database. If not, import will return an error indicating that some required users do not exist in the target database.

You can use FROMUSER and TOUSER to change the owners of objects. For example, if you specify `FROMUSER=dcranney,jfee TOUSER=smith, williams`, 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 does not have to have users `dcranney` and `jfee`, but must have users `smith` and `williams`.

After this statement successfully executes, all tablespaces in the set being copied remain in read-only mode. You should check the import logs to ensure no error has occurred. At this point, you can issue the ALTER TABLESPACE...READ WRITE statement to place the new tablespaces in read-write mode.

When dealing with a large number of datafiles, specifying the list of datafile names in the command line can be a laborious process; it may even exceed the command line limit. In this situation, you may use an import parameter file. For example, one of the commands in this step is equivalent to the following:

```
IMP PARFILE='par.f'
```

The file `par.f` contains the following:

```
TRANSPORT_TABLESPACE=y
DATAFILES=/db/staging.f
TABLESPACES=jan
TT_OWNERS=smith
```

To transport a tablespace between databases, both the source and target database must be running Oracle8i, with the init.ora compatibility parameter set to 8.1.

Object Behaviors

Most objects, whether data in a tablespace or structural information associated with the tablespace, behave normally after being transported to a different database. However, the following objects are exceptions:

- ROWIDs
- REFs
- Privileges
- Partitioned Tables
- Objects

- Advanced Queues
- Indexes
- Triggers
- Snapshots / Replication

ROWIDs

When a database contains tablespaces that have been plugged in (from other databases), the ROWIDs in that database are no longer unique. A ROWID is guaranteed unique only within a table.

REFs

REFs are not checked when Oracle determines if a set of tablespaces is self-contained. As a result, a plugged-in tablespace may contain dangling REFs. Any query following dangling REFs returns a user error.

Privileges

Privileges are transported if you specify GRANTS=y during export. During import, some grants may fail. For example, the user being granted a certain right may not exist, or a role being granted a particular right may not exist.

Partitioned Tables

You cannot move a partitioned table via transportable tablespaces when only a subset of the partitioned table is contained in the set of tablespaces. You must ensure that all partitions in a table are in the tablespace set, or exchange the partitions into tables before copying the tablespace set. However, you should note that exchanging partitions with tables invalidates the global index of the partitioned table.

At the target database, you can exchange the tables back into partitions if there is already a partitioned table that exactly matches the column in the target database. If all partitions of that table come from the same foreign database, the exchange operation is guaranteed to succeed. If they do not, in rare cases, the exchange operation may return an error indicating that there is a data object number conflict.

If you receive a data object conflict number error when exchanging tables back into partitions, you can move the offending partition using the ALTER TABLE MOVE PARTITION statement. After doing so, retry the exchange operation.

If you specify the WITHOUT VALIDATION option of the exchange statement, the statement will return immediately because it only manipulates structural information. Moving partitions, however, may be slow because the data in the partition can be copied. See "Transporting and Attaching Partitions for Data Warehousing: Example" on page 9-27 for an example using partitioned tables.

Objects

A transportable tablespace set can contain:

- tables
- indexes
- bitmap indexes
- index-organized tables
- LOBs
- nested tables
- varrays
- tables with user-defined type columns

If the tablespace set contains a pointer to a BFILE, you must move the BFILE and set the directory correctly in the target database.

Advanced Queues

You can use transportable tablespaces to move or copy Oracle advanced queues, as long as these queues are not 8.0-compatible queues with multiple recipients. After a queue is transported to a target database, the queue is initially disabled. After making the transported tablespaces read-write in the target database, you can enable the queue by starting it up via the built-in PL/SQL routine `dbms_aqadm.start_queue()`.

Indexes

You can transport regular indexes and bitmap indexes. When the transportable set fully contains a partitioned table, you can also transport the global index of the partitioned table.

Function-based indexes and domain indexes are not supported. If they exist in a tablespace, you must drop them before you can transport the tablespace.

Triggers

Triggers are exported without a validity check. In other words, Oracle does not verify that the trigger refers only to objects within the transportable set. Invalid triggers will cause a compilation error during the subsequent import.

Snapshots/Replication

Transporting snapshot or replication structural information is not supported. If a table in the tablespace you want to transport is replicated, you must drop the replication structural information and convert the table into a normal table before you can transport the tablespace.

Transporting and Attaching Partitions for Data Warehousing: Example

Typical enterprise data warehouses contain one or more large fact tables. These fact tables may be partitioned by date, making the enterprise data warehouse a historical database. You can build indexes to speed up star queries. In fact, 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's worth 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 wish 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 indexes 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; 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 option; otherwise the statement will go 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 will always succeed. In general, however, if data in a partitioned table comes from different databases, it's possible that the exchange operation may fail. For example, if the `jan98` partition of `sales` did not come from the same staging database, the above 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 may 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 in to an existing database without having to copy the datafiles from the CD to disk storage. For example, suppose on an 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:

```
IMP TRANSPORT_TABLESPACE=y DATAFILES='D:\catalog.f' FILE='D:\expdat.dmp'
```

You can remove the CD while the database is still up. Subsequent queries to the tablespace will return an error indicating that Oracle cannot open the datafiles on the CD. However, operations to other parts of the datafile 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 for a read-only tablespace. If you shut down and restart the database, Oracle will indicate that it cannot find the removed datafile and will 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, Oracle 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 tablespaces into each of the databases you wish 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 via several ways. You may use either a clustered file system or raw disk, as that is required by Oracle Parallel Server. Because Oracle will only read these type of datafiles on shared disk, you can also use NFS. Be aware, however, that if a user queries the shared tablespace while NFS is down, the database may hang until the NFS operation times out.

Later, you can drop the read-only tablespace in some of the databases. Doing so will not modify the datafiles for the tablespace; thus the drop operation will not corrupt the tablespace. Do not make the tablespace read-write unless only one database is mounting the tablespace.

Archive Historical Data via 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 via the transportable tablespace procedures described in this chapter.

See Also: For more details, see the *Oracle8i Backup and Recovery Guide.*

Using Transportable Tablespaces to Perform TSPITR

You can use transportable tablespaces to perform tablespace point-in-time recovery (TSPITR).

See Also: For information about how to perform TSPITR using transportable tablespaces, see the *Oracle8i Backup and Recovery Guide.*

Viewing Information About Tablespaces

The following data dictionary views provide useful information about tablespaces of a database:

- USER_EXTENTS, DBA_EXTENTS
- USER_SEGMENTS, DBA_SEGMENTS
- USER_FREE_SPACE, DBA_FREE_SPACE
- DBA_USERS
- DBA_TS_QUOTAS
- USER_TABLESPACES, DBA_TABLESPACES
- DBA_DATA_FILES
- V$DATAFILE

The following examples illustrate how to use the views not already illustrated in other chapters of this manual. They assume you are using a database that contains two tablespaces, SYSTEM and USERS. USERS is made up of two files, FILE1 (100MB) and FILE2 (200MB); the tablespace has been taken offline normally.

Listing Tablespaces and Default Storage Parameters: Example

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 sys.dba_tablespaces;

TABLESPACE  INITIAL_EXT  NEXT_EXT  MIN_EXT   MAX_EXT    PCT_INCREASE
----------  -----------  --------  -------   -------    ------------
SYSTEM      10240000     10240000        1      99            50
USERS       10240000     10240000        1      99            50
```

Listing the Datafiles and Associated Tablespaces of a Database: Example

To list the names, sizes, and associated tablespaces of a database, enter the following query on the DBA_DATA_FILES view:

```
SELECT  file_name, bytes, tablespace_name
   FROM sys.dba_data_files;

FILE_NAME    BYTES      TABLESPACE_NAME
------------ ---------- -------------------
filename1    10240000   SYSTEM
filename2    10240000   USERS
filename3    20480000   USERS
```

Listing the Free Space (Extents) of Each Tablespace: Example

To see the amount of space available in the free extents of 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 sys.dba_free_space
WHERE tablespace_name = 'SYSTEM'
GROUP BY tablespace_name, file_id;


TABLESPACE    FILE_ID  PIECES   MAXIMUM    MINIMUM  AVERAGE    TOTAL
----------    -------  ------   -------    -------  -------   ------
SYSTEM              1       2      2928        115   1521.5    3043
```

TOTAL shows the amount of free space in each tablespace, PIECES shows the amount of fragmentation in the datafiles of the tablespace, and MAXIMUM shows the largest contiguous area of space. 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.

10

Managing Datafiles

This chapter describes the various aspects of datafile management, and includes the following topics:

- Guidelines for Managing Datafiles
- Creating and Adding Datafiles to a Tablespace
- Changing a Datafile's Size
- Altering Datafile Availability
- Renaming and Relocating Datafiles
- Verifying Data Blocks in Datafiles
- Viewing Information About Datafiles

See Also: Datafiles can also be created as part of database recovery from a media failure. For more information, see the *Oracle8i Backup and Recovery Guide.*

Guidelines for Managing Datafiles

This section describes aspects of managing datafiles, and includes the following topics:

- Determine the Number of Datafiles
- Set the Size of Datafiles
- Place Datafiles Appropriately
- Store Datafiles Separate From Redo Log Files

Every datafile has two associated file numbers: an *absolute file number* and a *relative file number*.

An absolute file number uniquely identifies a datafile in the database. Prior to Oracle8, the absolute file number was referred to as simply the "file number."

A relative file number 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 will differ from the absolute file number. You can locate relative file numbers in many data dictionary views.

Determine the Number of Datafiles

At least one datafile is required for the SYSTEM tablespace of a database; a small system might have a single datafile. In general, keeping a few large datafiles is preferable to many small datafiles, because you can keep fewer files open at the same time.

You can add datafiles to tablespaces, subject to the following operating system-specific datafile limits:

operating system limit	Each operating system sets a limit on the maximum number of open files per process. Regardless of all other limits, more datafiles cannot be created when the operating system limit of open files is reached.
Oracle system limit	Oracle imposes a maximum limit on the number of datafiles for any Oracle database opened by any instance. This limit is port-specific.

control file upper bound	When you issue CREATE DATABASE or CREATE CONTROLFILE statements, the MAXDATAFILES parameter specifies an initial size of the datafile portion of the control file. Later, if you add a file whose number exceeds MAXDATAFILES but is less than or equal to DB_FILES, the control file automatically expands to allow the datafile portion to accommodate more files.
instance or SGA upper bound	When starting an Oracle8 instance, the database's parameter file indicates the amount of SGA space to reserve for datafile information; the maximum number of datafiles is controlled by the DB_FILES parameter. This limit applies only for the life of the instance. **Note:** The default value of DB_FILES is operating system specific. With the Oracle Parallel Server, all instances must set the instance datafile upper bound to the same value.

When determining a value for DB_FILES, take the following into consideration:

- If the value of DB_FILES is too low, you will be unable to 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.

Theoretically, an Oracle database can have an unlimited number of datafiles. Nevertheless, you should consider the following when determining the number of datafiles:

- Performance is better with a small number of datafiles rather than a large number of small datafiles. Large files also increase the granularity of a recoverable unit.
- Operating systems often impose a limit on the number of files a process can open simultaneously. Oracle's DBW0 process can open all online datafiles. Oracle is also 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.

Oracle allows more datafiles in the database than 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.

The operating system specific limit on the maximum number of datafiles allowed in a tablespace is typically 1023 files.

See Also: For more information on operating system limits, see your operating system-specific Oracle documentation.

For information about Parallel Server operating system limits, see *Oracle8i Parallel Server Concepts and Administration*.

For more information about MAXDATAFILES, see the *Oracle8i SQL Reference*.

Set the Size of Datafiles

The first datafile (in the original SYSTEM tablespace) must be at least 7M to contain the initial data dictionary and rollback segment. If you install other Oracle products, they may require additional space in the SYSTEM tablespace (for online help, for example); see the installation instructions for these products.

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, it might be helpful to store table data in a tablespace on one disk drive, and index data in a tablespace on another disk drive. This way, when users query table information, both disk drives can work simultaneously, retrieving table and index data at the same time.

Store Datafiles Separate From Redo Log Files

Datafiles should not be stored on the same disk drive that stores the database's 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 and Adding Datafiles to a Tablespace

You can create and add datafiles to a tablespace to increase the total amount of disk space allocated for the tablespace, and consequently the database.

Ideally, when creating a tablespace DBAs should estimate the potential size of the database objects and add sufficient files or devices. Doing so ensures that data is spread evenly across all devices.

To add datafiles to a tablespace, use either the Add Datafile dialog box of Enterprise Manager / GUI, or the SQL command ALTER TABLESPACE. You must have the ALTER TABLESPACE system privilege to add datafiles to a tablespace.

The following statement creates a new datafile for the RB_SEGS tablespace:

```
ALTER TABLESPACE rb_segs
   ADD DATAFILE 'filename1' SIZE 1M;
```

If you add new datafiles to a tablespace and do not fully specify the filenames, Oracle creates the datafiles in the default directory of the database server. Unless you want to reuse existing files, make sure the new filenames do not conflict with other files; the old files that have been previously dropped will be overwritten.

Changing a Datafile's Size

This section describes the various ways to alter the size of a datafile, and includes 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 files increase in specified increments up to a specified maximum.

Setting your datafiles to extend automatically results in the following:

- reduces the need for immediate intervention when a tablespace runs out of space
- ensures applications will not halt because of failures to allocate extents

To find out if a datafile is auto-extensible, query the DBA_DATA_FILES view and examine the AUTOEXTENSIBLE column.

You can specify automatic file extension when you create datafiles via the following SQL commands:

- CREATE DATABASE
- CREATE TABLESPACE
- ALTER TABLESPACE

You can enable or disable automatic file extension for existing datafiles, or manually resize a datafile using the SQL statement ALTER DATABASE.

The following example enables automatic extension for a datafile, FILENAME2, added to the USERS tablespace:

```
ALTER TABLESPACE users
    ADD DATAFILE 'filename2' 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 automatic extension for the datafile FILENAME2:

```
ALTER DATABASE DATAFILE 'filename2'
    AUTOEXTEND OFF;
```

See Also: For more information about the SQL statements for creating or altering datafiles, see the *Oracle8i SQL Reference.*

Manually Resizing a Datafile

You can manually increase or decrease the size of a datafile using the ALTER DATABASE command.

Because you can change the sizes of datafiles, you can 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.

Manually reducing the sizes of datafiles allows you to reclaim unused space in the database. This is useful for correcting errors in estimates of space requirements.

In this example, assume that the datafile FILENAME2 has extended up to 250M. However, because its tablespace now stores smaller objects, the datafile can be reduced in size.

The following command decreases the size of datafile FILENAME2:

```
ALTER DATABASE DATAFILE 'filename2'
   RESIZE 100M;
```

> **Note:** It is not always possible to decrease the size of a file to a specific value.

See Also: For more information about the implications resizing files has for downgrading, see *Oracle8i Migration*.

For more information about the ALTER DATABASE statement, see the *Oracle8i SQL Reference*.

Altering Datafile Availability

This section describes ways to alter datafile availability, and includes the following topics:

- Bringing Datafiles Online in ARCHIVELOG Mode
- Taking Datafiles Offline in NOARCHIVELOG Mode

In very rare situations, you might need to bring specific datafiles online (make them available) or take specific files offline (make them unavailable). For example, when Oracle has problems writing to a datafile, it can automatically take the datafile offline. You might need to take the damaged datafile offline or bring it online manually.

> **Note:** You can make all datafiles in a tablespace, other than the files in the SYSTEM tablespace, temporarily unavailable by taking the tablespace offline. You *must* leave these files in the tablespace to bring the tablespace back online.

Offline datafiles cannot be accessed. Bringing online a datafile in a read-only tablespace makes the file readable. No one can write to the file unless its associated tablespace is returned to the read-write state. The files of a read-only tablespace can

independently be taken online or offline using the DATAFILE option of the ALTER DATABASE command.

To bring a datafile online or take it offline, in either archiving mode, you must have the ALTER DATABASE system privilege. You can perform these operations only when the database is open in exclusive mode.

Bringing Datafiles Online in ARCHIVELOG Mode

To bring an individual datafile online, issue the SQL statement ALTER DATABASE and include the DATAFILE parameter.

> **Note:** To use this option 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.

The following statement brings the specified datafile online:

```
ALTER DATABASE DATAFILE 'filename' ONLINE;
```

See Also: For more information about bringing datafiles online during media recovery, see the *Oracle8i Backup and Recovery Guide*.

Taking Datafiles Offline in NOARCHIVELOG Mode

To take a datafile offline when the database is in NOARCHIVELOG mode, use the ALTER DATABASE command with the DATAFILE parameter and the OFFLINE DROP option. This allows you to take the datafile offline and drop it immediately. It is useful, for example, if the datafile contains only data from temporary segments and has not been backed up and the database is in NOARCHIVELOG mode.

The following statement brings the specified datafile offline:

```
ALTER DATABASE DATAFILE 'filename' OFFLINE DROP;
```

Renaming and Relocating Datafiles

This section describes the various aspects of renaming and relocating datafiles, and includes the following topics:

- Renaming and Relocating Datafiles for a Single Tablespace
- Renaming and Relocating Datafiles for Multiple Tablespaces

You can rename datafiles to change either their names or locations. Oracle provides options to make the following changes:

- Rename and relocate datafiles in a single offline tablespace (for example, FILENAME1 and FILENAME2 in TBSP1) while the rest of the database is open.
- Rename and relocate datafiles in several tablespaces simultaneously (for example, FILE1 in TBSP1 and FILE2 in TBSP2) while the database is mounted but closed.

> **Note:** To rename or relocate datafiles of the SYSTEM tablespace, you must use the second option, because you cannot take the SYSTEM tablespace offline.

Renaming and relocating datafiles with these procedures only change the pointers to the datafiles, as recorded in the database's control file; it does not physically rename any operating system files, nor does it copy files at the operating system level. Therefore, renaming and relocating datafiles involve several steps. Read the steps and examples carefully before performing these procedures.

You must have the ALTER TABLESPACE system privilege to rename datafiles of a single tablespace.

Renaming and Relocating Datafiles for a Single Tablespace

To Rename or Relocate Datafiles from a Single Tablespace

1. Take the non-SYSTEM tablespace that contains the datafiles offline.
2. Copy the datafiles to the new location or new names using operating system commands.
3. Make sure that the new, fully specified filenames are different from the old filenames.

4. Use the SQL statement ALTER TABLESPACE with the RENAME DATAFILE option to change the filenames within the database.

For example, the following statement renames the datafiles FILENAME1 and FILENAME2 to FILENAME3 and FILENAME4, respectively:

```
ALTER TABLESPACE users
  RENAME DATAFILE 'filename1', 'filename2'
      TO 'filename3', 'filename4';
```

The new file must already exist; this command does not create a file. Also, always provide complete filenames (including their paths) to properly identify the old and new datafiles. In particular, specify the old filename exactly as it appears in the DBA_DATA_FILES view of the data dictionary.

Renaming and Relocating Datafiles for Multiple Tablespaces

You can rename and relocate datafiles of one or more tablespaces using the SQL command ALTER DATABASE with the RENAME FILE option. This option is the only choice if you want to rename or relocate datafiles of several tablespaces in one operation, or rename or relocate datafiles of the SYSTEM tablespace. If the database must remain open, consider instead the procedure outlined in the previous section.

To rename datafiles of several tablespaces in one operation or to rename datafiles of the SYSTEM tablespace, you must have the ALTER DATABASE system privilege.

1. Ensure that the database is mounted but closed.
2. Copy the datafiles to be renamed to their new locations and new names, using operating system commands.
3. Make sure the new copies of the datafiles have different fully specified filenames from the datafiles currently in use.
4. Use the SQL statement ALTER DATABASE to rename the file pointers in the database's control file.

For example, the following statement renames the datafiles FILENAME 1 and FILENAME2 to FILENAME3 and FILENAME4, respectively:

```
ALTER DATABASE
  RENAME FILE 'filename1', 'filename2'
    TO 'filename3', 'filename4';
```

The new file must already exist; this command does not create a file. Also, always provide complete filenames (including their paths) to properly identify the old and

new datafiles. In particular, specify the old filename exactly as it appears in the DBA_DATA_FILES view of the data dictionary.

Relocating Datafiles: Example

For this example, assume the following conditions:

- An open database has a tablespace named USERS that is made up of datafiles located on the same disk of a computer.
- The datafiles of the USERS tablespace are to be relocated to a different disk drive.
- You are currently connected with administrator privileges to the open database while using Enterprise Manager.

To Relocate Datafiles

1. Identify the datafile names of interest.

   The following query of the data dictionary view DBA_DATA_FILES lists the datafile names and respective sizes (in bytes) of the USERS tablespace:

```
SELECT file_name, bytes FROM sys.dba_data_files
   WHERE tablespace_name = 'USERS';
FILE_NAME         BYTES
--------------------------
FILENAME1         102400000
FILENAME2         102400000
```

   Here, FILENAME1 and FILENAME2 are two fully specified filenames, each 1MB in size.

2. Back up the database.

   Before making any structural changes to a database, such as renaming and relocating the datafiles of one or more tablespaces, always completely back up the database.

3. Take the tablespace containing the datafile offline, or shut down the database and restart and mount it, leaving it closed. Either option closes the datafiles of the tablespace.

4. Copy the datafiles to their new locations using operating system commands. For this example, the existing files FILENAME1 and FILENAME2 are copied to FILENAME3 and FILENAME4.

   Note: You can execute an operating system command to copy a file by using the HOST command.

5. Rename the datafiles within Oracle.

   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 FILENAME1 and FILENAME2 to FILENAME3 and FILENAME4, respectively.

   If the tablespace is offline but the database is open, use the ALTER TABLESPACE...RENAME DATAFILE statement. If the database is mounted but closed, use the ALTER DATABASE...RENAME FILE statement.

6. Bring the tablespace online, or shut down and restart the database.

   If the USERS tablespace is offline and the database is open, bring the tablespace back online. If the database is mounted but closed, open the database.

7. Back up the database. After making any structural changes to a database, always perform an immediate and complete backup.

See Also: For more information about the DBA_DATA_FILES data dictionary view, see the *Oracle8i Reference*.

For more information about taking a tablespace offline, see "Taking Tablespaces Offline" on page 9-10.

Verifying Data Blocks in Datafiles

If you want to configure Oracle to use checksums to verify data blocks, set the initialization parameter DB_BLOCK_CHECKSUM to TRUE. The value of this parameter can be changed dynamically, or set in the initialization parameter file. The default value of DB_BLOCK_CHECKSUM is FALSE.

When you enable block checking, Oracle computes a checksum for each block written to disk. Checksums are computed for all data blocks, including temporary blocks.

The DBW0 process calculates the checksum for each block and stores it in the block's header. Checksums are also computed by the direct loader.

The next time Oracle reads a data block, it uses the checksum to detect corruption in the block. If a corruption is detected, Oracle returns message ORA-01578 and writes information about the corruption to a trace file.

WARNING: Setting DB_BLOCK_CHECKSUM to TRUE can cause performance overhead. Set this parameter to TRUE only under the advice of Oracle Support personnel to diagnose data corruption problems.

Viewing Information About Datafiles

The following data dictionary views provide useful information about the datafiles of a database:

- USER_EXTENTS, DBA_EXTENTS
- USER_SEGMENTS, DBA_SEGMENTS
- USER_FREE_SPACE, DBA_FREE_SPACE
- DBA_USERS
- DBA_TS_QUOTAS
- USER_TABLESPACES, DBA_TABLESPACES
- DBA_DATA_FILES
- V$DATAFILE

The following example illustrates how to use a view not already illustrated in other chapters of this manual. Assume you are using a database that contains two tablespaces, SYSTEM and USERS. USERS is made up of two files, FILE1 (100MB) and FILE2 (200MB); the tablespace has been taken offline normally. Here, you query V$DATAFILE to view status information about datafiles of a database:

```
SELECT name,
    file#,
    status,
    checkpoint_change# "CHECKPOINT"   FROM   v$datafile;

NAME                                 FILE#     STATUS       CHECKPOINT
-------------------------------        -----     -------       ----------
filename1                             1         SYSTEM       3839
filename2                             2         OFFLINE      3782
filename3                             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 written for a datafile's most recent checkpoint.

11

Using the Database Resource Manager

This chapter describes how to use the Database Resource Manager and includes the following topics:

- Using Database Resource Manager Packages
- Database Resource Manager Views

Introduction

Typically, when database resource allocation decisions are left to the operating system (OS), you may encounter the following problems:

- Excessive overhead

  Excessive overhead results from OS context switching between Oracle servers when the number of servers is high.

- Inefficient scheduling

  The OS de-schedules Oracle servers while they hold latches, which is inefficient.

- Poor resource partitioning

  The OS fails to partition CPU resources appropriately among tasks of varying importance.

- Inability to manage database-specific resources, such as parallel slaves and active sessions

Oracle's Database Resource Manager allocates resources based on a resource plan that is specified by database administrators. Database Resource Manager ultimately offers you more control over resource management decisions and addresses the problems caused by inefficient OS scheduling.

Administrators use the basic elements of Database Resource Manager described in Table 11–1.

Table 11–1 Database Resource Manager Elements

Element	Description
resource consumer group	user sessions grouped together based on resource processing requirements
resource plan	contains directives that specify which resources are allocated to resource consumer groups
resource allocation method	the method / policy used by Database Resource Manager when allocating for a particular resource; used by resource consumer groups and resource plans
resource plan directive	used by administrators to associate resource consumer groups with particular plans and partition resources among resource consumer groups

See Also: For detailed conceptual information about the Database Resource Manager, see *Oracle8i Concepts.*

Using Database Resource Manager Packages

To create resource plans and resource consumer groups, use the following packages:

- DBMS_RESOURCE_MANAGER
- DBMS_RESOURCE_MANAGER_PRIVS

Using the DBMS_RESOURCE_MANAGER Package

Use the DBMS_RESOURCE_MANAGER package to maintain resource plans, resource consumer groups, and plan directives. You can also use this package to group together changes to the plan schema.

You must have the SYSTEM privilege to administer the Database Resource Manager. Typically, administrators have this SYSTEM privilege with the ADMIN option. Following are procedures that grant and revoke this SYSTEM privilege.

```
grant_system_privilege(grantee_name in varchar2,admin_option in boolean)

revoke_system_privilege (revokee_name in varchar2)
```

Administering Resource Plans

Note: You must create a pending area before creating any Resource Manager objects. For more details see "Creating and Administering the Pending Area" on page 11-5.

You can use the following procedures to create, update, or delete resource plans:

```
create_plan(plan in varchar2, comment in varchar2,
   cpu_mth in varchar2 DEFAULT 'EMPHASIS',
   max_active_sess_target_mth in varchar2 DEFAULT
   'MAX_ACTIVE_SESS_ABSOLUTE',
   parallel_degree_limit_mth in varchar2 DEFAULT
   'PARALLEL_DEGREE_LIMIT_ABSOLUTE')
update_plan(plan in varchar2, new_comment in varchar2)
   DEFAULT NULL, new_cpu_mth in varchar2
   DEFAULT NULL, new_max_active_sess_target_mth in
   varchar2 DEFAULT NULL,
```

```
   new_parallel_degree_limit_mth in varchar2
   DEFAULT NULL)
delete_plan(plan in varchar2)
delete_plan_cascade(plan in varchar2)
```

The `delete_plan` procedure deletes the specified plan as well as all the plan directives it refers to. 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.

If you do not specify the arguments to `update_plan` procedure, they remain unchanged in the data dictionary.

If you wish to use a default resource allocation method, you need not specify it when creating or updating a plan. The method defaults are:

- cpu_method ='EMPHASIS'
- parallel_degree_limit_mth ='PARALLEL_DEGREE_LIMIT_ABSOLUTE'

Administering Resource Consumer Groups

You can use the following procedures to create, update, or delete resource consumer groups:

```
create_consumer_group(consumer_group in varchar2,
   comment in varchar2, cpu_mth in varchar2
   DEFAULT 'ROUND-ROBIN')
update_consumer_group(consumer_group in varchar2,
   new_comment in varchar2 DEFAULT NULL,
   new_cpu_mth in varchar2 DEFAULT NULL)
delete_consumer_group(consumer_group in varchar2)
```

You need not specify the `cpu_mth` parameter if you wish to use the default CPU method, which is ROUND-ROBIN.

If you do not specify the arguments for the `update_consumer_group` procedure, they remain unchanged in the data dictionary.

Administering Resource Plan Directives

You can use the following procedures to create, update, or delete resource plan directives:

```
create_plan_directive(plan in varchar2, group_or_subplan
   in varchar2, comment in varrchar2, cpu_p1 in number
DEFAULT NULL, cpu_p2 in number
```

```
DEFAULT NULL, cpu_p3 in number
DEFAULT NULL, cpu_p4 in number
DEFAULT NULL, cpu_p5 in number
DEFAULT NULL, cpu_p6 in number
DEFAULT NULL, cpu_p7 in number
DEFAULT NULL, cpu_p8 in number
DEFAULT NULL, max_active_sess_target_p1 in number
DEFAULT NULL, parallel_degree_limit_p1 in number DEFAULT NULL)
update_plan_directive(plan in varchar2, group_or_subplan
   in varchar2, new_comment in varchar2
DEFAULT NULL, new_cpu_p1 in number
DEFAULT NULL, new_cpu_p2 in number
DEFAULT NULL, new_cpu_p3 in number DEFAULT NULL, new_cpu_p4 in number
DEFAULT NULL, new_cpu_p5 in number DEFAULT NULL, new_cpu_p6 in number
DEFAULT NULL, new_cpu_p7 in number DEFAULT NULL, new_cpu_p8 in number
DEFAULT NULL, max_active_sess_target_p1 in number
DEFAULT NULL, new_parallel_degree_limit_p1 in number
DEFAULT NULL)
delete_plan_directive(plan in varchar2, group_or_subplan
  in varchar2)
```

All parameters default to NULL.

If you do not specify the arguments for the `update_plan_directive` procedure, they remain unchanged in the data dictionary.

Creating and Administering the Pending Area

All changes to the plan schema can be done within a *pending area*, which is a "scratch" area for plan schema changes. You must create this pending area, make changes as necessary and submit the changes (validation is optional).

You can use the following procedures to create, validate, and submit pending changes for the Database Resource Manager:

```
dbms_resource_manager.create_pending_area

dbms_resource_manager.validate_pending_area

dbms_resource_manager.clear_pending_area

dbms_resource_manager.submit_pending_area
```

> **Note:** The changes come into effect and become active only if the `submit_pending_area` procedure completes successfully.

You can also view the current schema containing your changes by selecting from the appropriate user views while the pending area is active. You can clear the pending area to abort the current changes any time as well. Call the `validate` procedure to check whether your changes are valid.

The changes made within the pending area must adhere to the following rules:

1. No plan schema may contain any loops.
2. All plan and / or 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 for the emphasis resource allocation method.
5. A plan that is currently being used as a top plan by an active instance cannot be deleted.
6. The plan directive parameter `parallel_degree_limit_p1` can appear only in plan directives that refer to resource consumer groups (not other resource plans).
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. All leaves of a top plan must be consumer resource groups; at the lowest level in a plan schema the plan directives must refer to consumer groups.
8. Plans and resource consumer groups may not have the same name.
9. There must be a plan directive for OTHER_GROUPS somewhere in an active plan schema. This ensures that a session not covered by the currently active plan is allocated resources as specified by the OTHER_GROUPS directive.

Database Resource Manager allows "orphan" resource consumer groups (resource consumer groups with no plan directives referring to them) because you may wish to create a resource consumer group that is not currently being used, but will be used in the future.

You will receive an error message if any of the above rules are broken when checked by the `validate` or `submit` procedures. You may then make changes to fix the problem(s) and reissue the `validate` or `submit` procedures. The

`submit_pending_area` clears the pending area after validating and committing the changes (if valid).

> **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`.

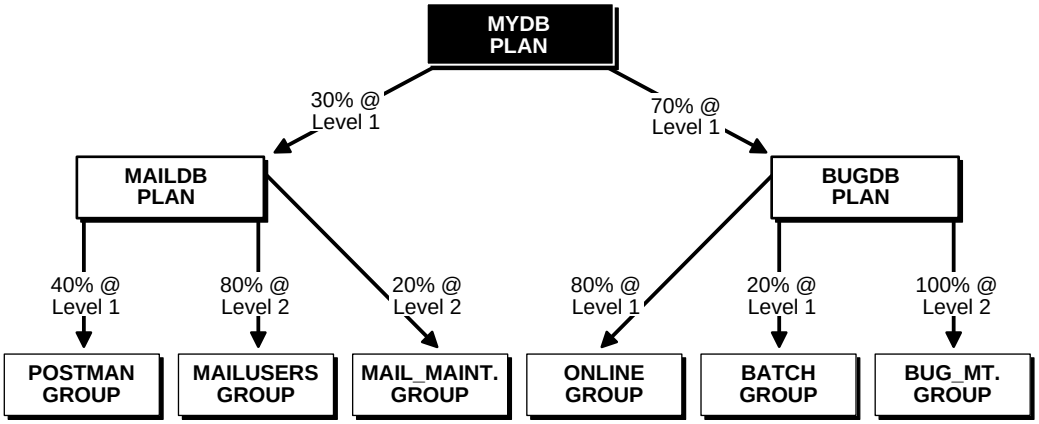
The following commands create a multi-level schema, and use the default plan and resource consumer group methods as illustrated in Figure 11–1:

```
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 bug users' sessions who run batch jobs');
dbms_resource_manager.create_consumer_group(consumer_group => 'Bug_Maintenance_group',
   comment => 'Resource consumer group/method for users' sessions who maintain
   the bug  db');
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 who maintain the mail
   db');
dbms_resource_manager.create_plan_directive(plan => 'BUGDB_PLAN', group_or_subplan =>
'Bug_Online_group',
   comment => 'online bug users' sessions at level 0', 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 0', 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 1', cpu_p1 => 0, cpu_p2 => 100,
parallel_degree_limit_p1 => 3);
```

```
dbms_resource_manager.create_plan_directive(plan => 'MAILDB_PLAN', group_or_subplan =>
'Mail_Postman_group',
   comment => 'mail postman at level 0', 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 1', 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 1', cpu_p1 => 0, cpu_p2 => 20,
   parallel_degree_limit_p1 => 2);
dbms_resource_manager.create_plan_directive(plan => 'MYDB_PLAN', group_or_subplan =>
'MAILDB_PLAN',
   comment=> 'all mail users' sessions at level 0', cpu_p1 => 30);
dbms_resource_manager.create_plan_directive(plan => 'MYDB_PLAN', group_or_subplan =>
'BUGDB_PLAN',
   comment => 'all bug users' sessions at level 0', 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 11–1 Multi-level Schema



Assigning Resource Consumer Groups to Users

In addition to providing the above procedures to maintain resource plans and resource consumer groups, the DATABASE_RESOURCE_MANAGER package also contains procedures to assign resource consumer groups to users. The following procedure sets the initial consumer group of a user:

```
set_initial_consumer_group(user in varchar2, consumer_group in varchar2)
```

The initial consumer group of a user is the consumer group to which any session created by that user initially belongs. You must grant the switch privilege directly to the user or PUBLIC before it can be the user's initial consumer group. The switch privilege for the initial consumer group cannot come from a role granted to that user (these semantics are similar to those for ALTER USER DEFAULT ROLE).

If you have not set the initial consumer group for a user, the user's initial consumer group will automatically be the consumer group DEFAULT_CONSUMER_GROUP. DEFAULT_CONSUMER_GROUP has switch privileges granted to PUBLIC; therefore, all users are automatically granted switch privilege for this 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 as their initial consumer group. All sessions belonging to a deleted consumer group will be switched to DEFAULT_CONSUMER_GROUP.

Changing Resource Consumer Groups

You can use the following procedure to change the resource consumer group of a specific session:

```
switch_consumer_group_for_sess(session_id in number, session_serial
   in number, consumer_group in varchar2)
```

You can use the following procedure to change the resource consumer group for all sessions with a given user id:

```
switch_consumer_group_for_user(user in varchar2, class in varchar2)
```

Both procedures also change the resource consumer group of any (PQ) slave sessions that are related to the top user session.

See Also: For information about views associated with Database Resource Manager, see the *Oracle8i Reference*.

The DBMS_RESOURCE_MANAGER_PRIVS Package

Use the DBMS_RESOURCE_MANAGER_PRIVS package to maintain privileges associated with resource consumer groups. The procedures in this package are executed with the privileges of the caller.

Granting Switch Privileges

To grant the privilege to switch to a consumer group, use the following procedure:

```
grant_switch_consumer_group(grantee_name in varchar2, consumer_group in varchar2,
   grant_option in boolean)
```

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 may also grant switch privileges for that consumer group to others.

Revoking Switch Privileges

To revoke the privilege to switch to resource consumer groups, use the following procedure:

```
revoke_switch_consumer_group(revokee_name in varchar2, consumer_group in varchar2)
```

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 a role's switch privileges to a consumer group, then any users who only had switch privilege for the consumer group via that role will not be able to subsequently switch to that consumer group.

If you revoke from PUBLIC switch privileges to a consumer group, then any users who could previously only use the consumer group via PUBLIC will not be able to subsequently switch to that consumer group.

Using the DBMS_SESSION Package to Change a User's Resource Consumer Groups

You can change your current resource consumer group by calling the following procedure in the DBMS_SESSION package:

```
switch_current_consumer_group(new_consumer_group in varchar2,
   old_consumer_group out varchar2, initial_group_on_error in boolean)
```

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. This procedure also returns the old consumer group to users, and can be used to switch back to the old consumer group later.

The parameter `initial_group_on_error` controls the behavior of the procedure in the event of an error; if the parameter is set to TRUE and an error occurs, the invoker's consumer group is set to his/her initial consumer group.

Database Resource Manager Views

The following dynamic performance table views are associated with Database Resource Manager:

- V$SESSION
- V$MYSESSION
- V$RSRC_CONSUMER_GROUP
- V$RSRC_PLAN
- V$RSRC_CONSUMER_GROUP_CPU_MTH
- V$RSRC_PLAN_CPU_MTH
- V$PARALLEL_DEGREE_LIMIT_MTH

The following static data dictionary views are associated with Database Resource Manager:

- DBA_RSRC_CONSUMER_GROUP_PRIVS
- DBA_RSRC_MANAGER_SYSTEM_PRIVS
- DBA_RSRC_CONSUMER_GROUPS
- DBA_RSRC_PLAN_DIRECTIVES
- DBA_RSRC_PLANS
- USER_RSRC_CONSUMER_GROUP_PRIVS
- USER_RSRC_MANAGER_SYSTEM_PRIVS
- DBA_USERS
- USERS_USERS

See Also: For detailed information about the contents of each of these views, see the *Oracle8i Reference.*

12

Guidelines for Managing Schema Objects

This chapter describes guidelines for managing schema objects, and includes the following topics:

- Managing Space in Data Blocks
- Setting Storage Parameters
- Deallocating Space
- Understanding Space Use of Datatypes

You should familiarize yourself with the concepts in this chapter before attempting to manage specific schema objects as described in Chapters 13–18.

Managing Space in Data Blocks

This section describes the various aspects of managing space in data blocks, and includes the following topics:

- The PCTFREE Parameter
- The PCTUSED Parameter
- Selecting Associated PCTUSED and PCTFREE Values

You can use the PCTFREE and PCTUSED parameters to make the following changes:

- increase the performance of writing and retrieving data
- decrease the amount of unused space in data blocks
- decrease the amount of row chaining between data blocks

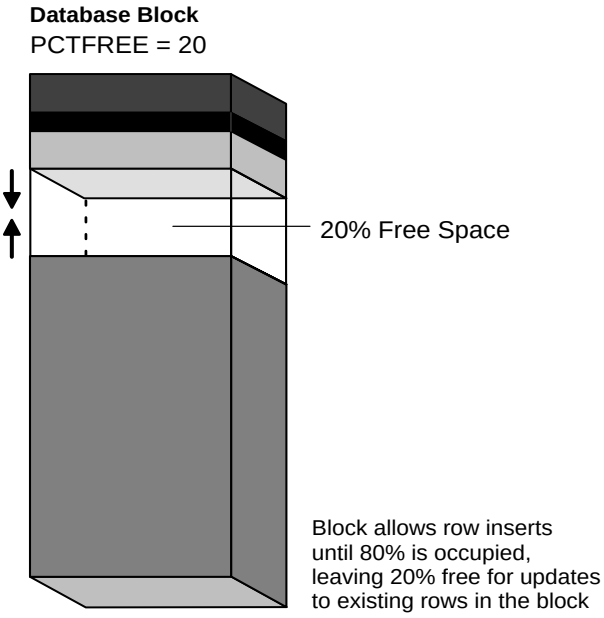
The PCTFREE Parameter

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 this table's data segment will be kept free and available for possible updates to the existing rows already within each block. Figure 12–1 illustrates PCTFREE.

Figure 12–1 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.

Specifying PCTFREE

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.

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 per block)

A small PCTFREE might be suitable, for example, for a segment that is rarely changed.

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 per block)
- may improve update performance, because Oracle does not need to chain row pieces as frequently, if ever

A large PCTFREE is suitable, for example, for segments that are frequently updated.

Ensure that you understand the nature of the 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.

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 chained among blocks.

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.

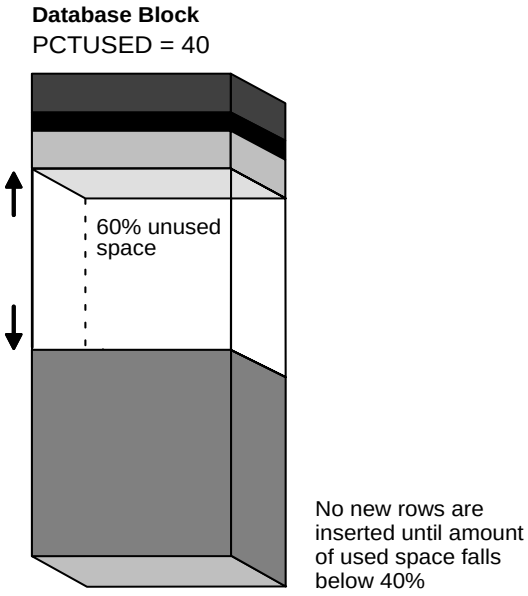
The PCTUSED Parameter

After a data block becomes full as determined by PCTFREE, Oracle 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, Oracle 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 this table's data segment 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 block's used space has previously reached PCTFREE). Figure 12–2 illustrates this.

Figure 12–2 PCTUSED



Specifying 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.

A smaller PCTUSED has the following effects:

- 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
- increases the unused space in a database

A larger PCTUSED has the following effects:

- improves space efficiency
- increases processing cost during INSERTs and UPDATEs

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 Oracle attempts to keep no more than PCTFREE free space, and processing costs are highest.
- Block overhead is not included in the computation of PCTUSED or PCTFREE.
- 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.

Examples of Choosing PCTFREE and PCTUSED Values

The following examples show how and why specific values for PCTFREE and PCTUSED are specified for tables.

Example 1	Scenario:	Common activity includes UPDATE statements that increase the size of the rows.
	Settings:	PCTFREE = 20
		PCTUSED = 40
Example 2	Scenario:	Most activity includes INSERT and DELETE statements, and UPDATE statements that do not increase the size of affected rows.
	Settings:	PCTFREE = 5
		PCTUSED = 60
	Explanation:	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.
Example 3	Scenario:	The table is very large; therefore, storage is a primary concern. Most activity includes read-only transactions.
	Settings:	PCTFREE = 5
		PCTUSED = 40
	Explanation:	PCTFREE is set to 5 because this is a large table and you want to completely fill each block.

Setting Storage Parameters

This section describes the storage parameters you can set for various data structures, and includes the following topics:

You can set storage parameters for the following types of logical storage structures:

- tablespaces (most defaults for any segment in the tablespace)
- tables, clusters, snapshots, and snapshot logs (data segments)
- indexes (index segments)
- rollback segments

Storage Parameters You Can Specify

Every database has default values for storage parameters. You can specify defaults for a tablespace, which override the system defaults to become the defaults for objects created in that tablespace only. Furthermore, you can specify storage settings for each individual object. The storage parameters you can set are:

INITIAL

The size, in bytes, of the first extent allocated when a segment is created.

Default: 5 data blocks
Minimum: 2 data blocks (rounded up)
Maximum: operating system specific

NEXT

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).

Default: 5 data blocks
Minimum: 1 data block
Maximum: operating system specific

MAXEXTENTS

The total number of extents, including the first, that can ever be allocated for the segment.

Default: dependent on the data block size and operating system
Minimum: 1 (extent)
Maximum: unlimited

MINEXTENTS

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.

Default: 1 (extent)
Minimum: 1 (extent)
Maximum: unlimited

PCTINCREASE

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.

Default: 50 (%)
Minimum: 0 (%)
Maximum: operating system specific

INITRANS

Specifies the number of DML transaction entries for which space should be initially reserved in the data block header. Space is reserved in the headers of all data blocks in the associated data or index segment.

The default value is 1 for tables and 2 for clusters and indexes.

MAXTRANS

As multiple transactions concurrently access the rows of the same data block, space is allocated for each DML transaction's 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. The MAXTRANS parameter limits the number of transaction entries that can concurrently use data in a data block. Therefore, you can limit the amount of free space that can be allocated for transaction entries in a data block using MAXTRANS.

The default value is an operating system-specific function of block size, not exceeding 255.

See Also: For specific details about storage parameters, see the *Oracle8i SQL Reference*.

Some defaults are operating system specific; see your operating system-specific Oracle documentation.

Setting INITRANS and MAXTRANS

Transaction entry settings for the data blocks allocated for a table, cluster, or index should be set individually for each object based on the following criteria:

- 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 (to eliminate the overhead of having to allocate transaction entry space, as required when the object is in use) and allowing a higher MAXTRANS so that no user has to wait to access necessary data blocks.

Setting Default Storage Parameters for Segments in a Tablespace

You can set default storage parameters for each tablespace of a database. Any storage parameter that you do not explicitly set when creating or subsequently altering a segment in a tablespace automatically is set to the corresponding default storage parameter for the tablespace in which the segment resides.

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 partitioned table (unless you specify them for the individual partition). If no storage parameters are specified for the partitioned table, then they are inherited from the tablespace.

When specifying MINEXTENTS at the tablespace level, any extent allocated in the tablespace is rounded to a multiple of the number of minimum extents. Basically, the number of extents is a multiple of the number of blocks.

Setting Storage Parameters for Data Segments

You can set the storage parameters for the data segment of a nonclustered table, snapshot, or snapshot log using the STORAGE clause of the CREATE or ALTER statement for tables, snapshots, or snapshot logs.

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 command, rather than the individual CREATE or ALTER commands that put tables and snapshots into the cluster. Storage parameters specified when creating or altering a *clustered* table or snapshot are ignored. The storage parameters set for the cluster override the table's storage parameters.

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 command. Storage parameters of an index segment created for the index used to enforce a primary key or unique key constraint can be set in the ENABLE clause of the CREATE TABLE or ALTER TABLE commands or the STORAGE clause of the ALTER INDEX command.

A PCTFREE setting for an index only has an effect when the index is created. You cannot specify PCTUSED for an index segment.

Setting Storage Parameters for LOB Segments

You can set storage parameters for LOB segments using the NOCACHE, NOLOGGING and PCTVERSION LOB storage parameters of the CREATE TABLE statement.

See Also: For a complete list of storage parameters for LOB segments, see the *Oracle8i SQL Reference*.

Changing Values for Storage Parameters

You can alter default storage parameters for tablespaces and specific storage parameters for individual segments if the current settings are incorrect. All 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.

The INITIAL and MINEXTENTS storage parameters cannot be altered for an existing table, cluster, index, or rollback segment. 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

The storage parameters in effect at a given time are determined by the following types of SQL statements, listed in order of precedence:

1. ALTER TABLE / CLUSTER / SNAPSHOT / SNAPSHOT LOG / INDEX / ROLLBACK SEGMENT statement
2. CREATE TABLE / CLUSTER / SNAPSHOT / SNAPSHOT LOG / CREATE INDEX / ROLLBACK SEGMENT statement
3. ALTER TABLESPACE statement
4. CREATE TABLESPACE statement
5. Oracle default values

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 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.

Storage Parameter Example

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 12–1 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	255 blocks or 522240 bytes

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 on.

Deallocating Space

This section describes aspects of deallocating unused space, and includes the following topics:

- Viewing the High Water Mark
- Issuing Space Deallocation Statements

It is not uncommon to allocate space to a segment, only to find out later that it is not being used. For example, you may set PCTINCREASE to a high value, which could create a large extent that is only partially used. Or you could explicitly overallocate space by issuing the ALTER TABLE...ALLOCATE EXTENT statement. If you find that you have unused or overallocated space, you can release it so that the unused space can be used by other segments.

Viewing the High Water Mark

Prior to deallocation, you can use the DBMS_SPACE package, which contains a procedure (UNUSED_SPACE) that returns information about the position of the high water mark and the amount of unused space in a segment.

Within a segment, the high water mark indicates the amount of used space. You cannot release space below the high water mark (even if there is no data in the space you wish to deallocate). However, if the segment is completely empty, you can release space using the TRUNCATE...DROP STORAGE statement.

Issuing Space Deallocation Statements

The following statements deallocate unused space in a segment (table, index or cluster). The KEEP clause is *optional*.

```
ALTER TABLE table DEALLOCATE UNUSED KEEP integer;
ALTER INDEX index DEALLOCATE UNUSED KEEP integer;
ALTER CLUSTER cluster DEALLOCATE UNUSED KEEP integer;
```

When you explicitly identify an amount of unused space to KEEP, this space is retained while the remaining unused space is deallocated. If the remaining number of extents becomes smaller than MINEXTENTS, the MINEXTENTS value changes to reflect the new number. If the initial extent becomes smaller, the INITIAL value changes to reflect the new size of the initial extent.

If you do not specify the KEEP clause, all unused space (everything above the high water mark) is deallocated, as long as the size of the initial extent and MINEXTENTS are preserved. Thus, even if the high water mark occurs within the MINEXTENTS boundary, MINEXTENTS remains and the initial extent size is not reduced.

See Also: For details on the syntax and options associated with deallocating unused space, see the *Oracle8i SQL Reference*.

You can verify that deallocated space is freed by looking at the DBA_FREE_SPACE view. For more information on this view, see the *Oracle8i Reference*.

Deallocating Space: Examples

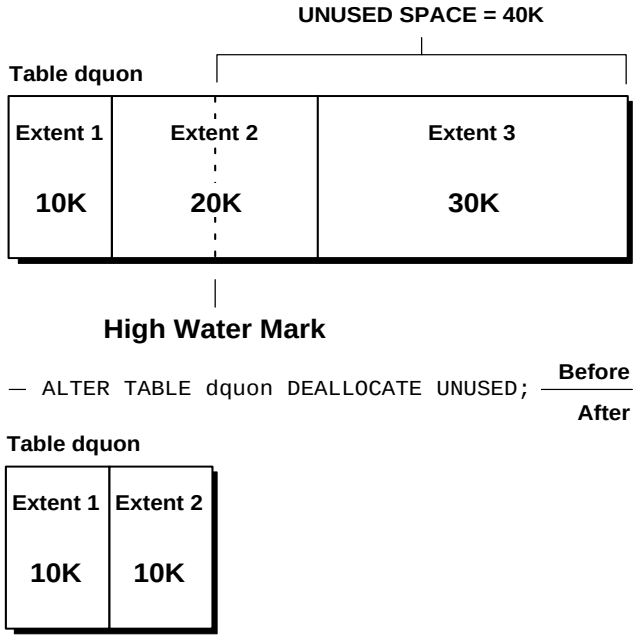
This section includes various space deallocation scenarios. Prior to reading it, you should familiarize yourself with the ALTER...DEALLOCATE UNUSED statements in the *Oracle8i Reference*.

Example 1

Table DQUON consists of three extents (see Figure 12–3). The first extent is 10K, the second is 20K, and the third is 30K. The high water mark is in the middle of the second extent, and there is 40K of unused space. The following statement deallocates all unused space, leaving table DQUON with two remaining extents. The third extent disappears, and the second extent size is 10K.

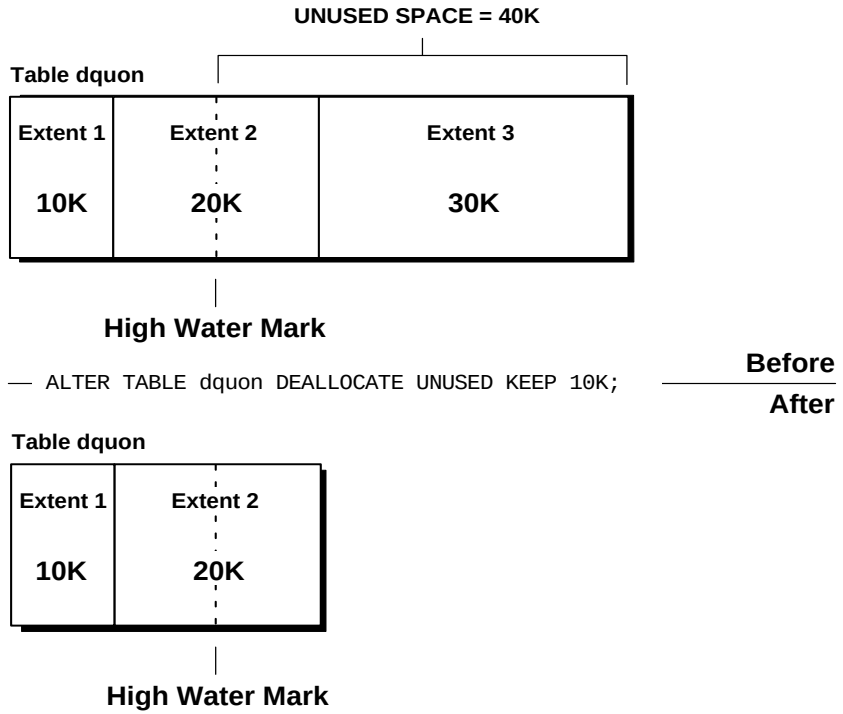
```
ALTER TABLE dquon DEALLOCATE UNUSED;
```

Figure 12–3 Deallocating All Unused Space



If you deallocate all unused space from DQUON and KEEP 10K (see Figure 12–4), the third extent is deallocated and the second extent remains in tact.

Figure 12–4 Deallocating Unused Space, KEEP 10K



If you deallocate all unused space from DQUON and KEEP 20K, the third extent is cut to 10K, and the size of the second extent remains the same.

```
ALTER TABLE dquon DEALLOCATE UNUSED KEEP 20K;
```

Example 2

When you issue the ALTER TABLE DQUON DEALLOCATE UNUSED statement, you completely deallocate the third extent, and the second extent is left with 10K. Note that the size of the next allocated extent defaults to the size of the last completely deallocated extent, which in this example, is 30K. However, if you can explicitly set the size of the next extent using the ALTER...STORAGE [NEXT] statement.

Example 3

To preserve the MINEXTENTS number of extents, DEALLOCATE can retain extents that were originally allocated to an instance (added below the high water mark), while deallocating extents that were originally allocated to the segment.

For example, table DQUON has a MINEXTENTS value of 2. Examples 1 and 2 still yield the same results. However, if the MINEXTENTS value is 3, then the ALTER TABLE DQUON DEALLOCATE UNUSED statement has no effect, while the ALTER TABLE DQUON DEALLOCATE UNUSED KEEP 10K statement removes the third extent and changes the value of MINEXTENTS to 2.

Understanding Space Use 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, as described below.

Character Datatypes

The CHAR and VARCHAR2 datatypes store alphanumeric data in strings of ASCII (American Standard Code for Information Interchange) or EBCDIC (Extended Binary Coded Decimal Interchange Code) values, depending on the character set used by the hardware that runs Oracle. Character datatypes can also store data using character sets supported by the National Language Support (NLS) feature of Oracle.

The CHAR datatype stores fixed-length character strings. When a table is created with a CHAR column, a column length (in bytes, not characters) between 1 and 255 can be specified for the CHAR column; the default is 1 byte. Extra blanks are used to fill remaining space in the column for values less than the column length.

The VARCHAR2 datatype stores variable-length character strings. When a table is created with a VARCHAR2 column, a maximum column length (in bytes, not characters) between 1 and 4000 is specified for the VARCHAR2 column. For each row, each value in the column is stored as a variable- length field. Extra blanks are not used to fill remaining space in the column.

Number Datatype

The NUMBER datatype stores fixed and floating point numbers. Positive numbers in the range 1×10^{-130} to $9.99...9 \times 10^{125}$ (with up to 38 significant digits), negative numbers in the range

You can optionally specify a *precision* (total number of digits) and *scale* (number of digits to the right of the decimal point) when defining a NUMBER column. Oracle guarantees portability of numbers with a precision equal to or less than 38 digits. You can specify a scale and no precision:

-1×10^{-130} to $-9.99..9 \times 10^{125}$ (with up to 38 significant digits), and zero. If precision is not specified, the column stores values as given. If no scale and no precision are specified:

```
column_name NUMBER (*, scale)
```

In this case, the precision is 38 and the specified scale is maintained.

DATE Datatype

The DATE datatype stores point-in-time values such as dates and times. Date data is stored in fixed length fields of seven bytes each.

LONG Datatype

Columns defined as LONG store variable length character data containing up to two gigabytes of information. LONG data is text data and is appropriately converted when moved between different character sets. LONG data cannot be indexed.

Note: You can convert LONG data to LOB data, which has fewer restrictions. For more information see the *Oracle8i Application Developer's Guide - Large Objects (LOBs).*

RAW and LONG RAW Datatypes

RAW is a variable-length datatype like the VARCHAR2 character datatype, except that Net8 (which connects user sessions to the instance) and the Import and Export utilities do not perform character conversion when transmitting RAW or LONG RAW data. In contrast, Net8 and Export/Import automatically convert CHAR, VARCHAR2, and LONG data between the database character set and the user session character set if the two character sets are different.

RAW data can be indexed; LONG RAW data cannot be indexed.

rowids, ROWID pseudocolumn and ROWID datatype

Every row in a nonclustered table of an Oracle database is assigned a unique rowid that corresponds to the physical address of a row's row piece (or the initial row piece if the row is chained among multiple row pieces).

Each table in an Oracle database has an internal pseudo-column named ROWID. This pseudocolumn is not evident when listing the structure of a table by executing a SELECT statement, or a DESCRIBE statement using SQL*Plus, but can be retrieved with a SQL query using the reserved word ROWID as a column name.

rowids use a binary representation of the physical address for each row selected. A rowid's VARCHAR2 hexadecimal representation is divided into three pieces: block.slot.file. Here, block is the data block within a file that contains the row, relative to its datafile; row is the row in the block; and file is the datafile that contains the row. A row's assigned ROWID remains unchanged usually. Exceptions occur when the row is exported and imported (using the Import and Export utilities). When a row is deleted from a table (and the encompassing transaction is committed), the deleted row's associated ROWID can be assigned to a row inserted in a subsequent transaction.

See Also: For more information about NLS and support for different character sets, see the *Oracle8i National Language Support Guide*.

For more information about datatypes, see the *Oracle8i SQL Reference*.

Summary of Oracle Datatypes

Table 12–2 summarizes important information about each Oracle datatype.

Table 12–2 Summary of Oracle Datatype Information

Datatype	Description	Column Length (bytes)
CHAR *(size)*	Fixed-length character data of length size.	Fixed for every row in the table (with trailing spaces); maximum size is 2000 bytes per row, default size is one byte per row. Consider the character set that is used before setting size. (Are you using a one or two byte character set?)
NCHAR *(size)*	Fixed-length character data of length *size* characters or bytes, depending on the choice of national character set.	Maximum *size* is determined by the number of bytes required to store each character, with an upper limit of 2000 bytes. Default and minimum *size* is one character or one byte, depending on the character set.

Table 12–2 Summary of Oracle Datatype Information (Cont.)

Datatype	Description	Column Length (bytes)
VARCHAR2 *(size)*	Variable-length character string having maximum length *size* bytes. A maximum size must be specified.	Maximum *size* is 4000, and minimum is 1. You must specify *size* for VARCHAR2.
NVARCHAR2 (size)	Variable-length character string having maximum length *size* characters or bytes, depending on the choice of national character set.	Maximum *size* is determined by the number of bytes required to store each character, with an upper limit of 4000 bytes. You must specify *size* for NVARCHAR2.
NUMBER (*p*, *s*)	Number having precision *p* and scale *s*. The precision *p* can range from 1 to 38. The scale *s* can range from -84 to 127.	Variable for each row. The maximum space required for a given column is 21 bytes per row.
DATE	Fixed-length date and time data, ranging from January 1, 4712 B.C. to December 31, 4712 A.D. Default format: DD-MON-YY.	Fixed at seven bytes for each row in the table.
LONG	Variable-length character data.	Variable for each row in the table up to 2^{31} -1 bytes, or two gigabytes per row.
RAW *(size)*	Raw binary data of length *size* bytes.	Variable for each row in the table, up to 2000 bytes per row. You must specify *size* for a RAW value.
LONG RAW	Variable-length raw binary data.	Variable for each row in the table, up to two gigabytes per row.
ROWID	Hex data representing unique row addresses. This datatype is primarily for values returned by the ROWID pseudocolumn.	Fixed at six bytes for each row in the table.

Table 12–2 Summary of Oracle Datatype Information (Cont.)

Datatype	Description	Column Length (bytes)
UROWID [(size)]	Hexadecimal string representing the logical address of a row of an index-organized table.	The optional *size* is the size of a column of type UROWID. The maximum size and default is 4000 bytes.
CHAR (*size*)	Fixed-length character data of length *size* bytes.	Maximum *size* is 2000 bytes. Default and minimum *size* is 1 byte.

13

Managing Partitioned Tables and Indexes

This chapter describes various aspects of managing partitioned tables and indexes, and includes the following sections:

What Are Partitioned Tables and Indexes?

Note: Before attempting to create a partitioned table or index or perform maintenance operations on any partition, review the information about partitioning in *Oracle8i Concepts.*

Today's 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 or indexes can be divided into a number of pieces, called *subpartitions,* which have 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) in an index share the same index options. Each partition (or subpartition) is stored in a separate segment and can have different physical attributes (such as PCTFREE, PCTUSED, INITRANS, MAXTRANS, TABLESPACE, and STORAGE).

Although you are not required to keep each table or index partition 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

See Also: For more detailed information on partitioning concepts and benefits, see *Oracle8i Concepts.*

Partitioning Methods

There are three partitioning methods:

- Range Partitioning
- Hash Partitioning
- Composite Partitioning

This section describes how to use each of these methods.

Using the Range Partitioning Method

You can use range partitioning to map rows to partitions based on ranges of column values. Range partitioning is defined by the partitioning specification for a table or index, and by the partitioning specifications for each individual partition.

The following example shows a table of four partitions (one for each quarter's sales); a row with SALE_YEAR=1998, SALE_MONTH=8 and SALE_DAY=18 has partitioning key (1998, 8, 18), belongs in the third partition, and is stored in tablespace TSC. A row with SALE_YEAR=1998, SALE_MONTH=8 and SALE_DAY=1 has partitioning key (1998, 8, 1), and also belongs in the third partition, stored in tablespace TSC.

```
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 (1998, 04, 01)
        TABLESPACE tsa,
      PARTITION sales_q2 VALUES LESS THAN (1998, 07, 01)
        TABLESPACE tsb,
      PARTITION sales_q3 VALUES LESS THAN (1998, 10, 01)
        TABLESPACE tsc,
      PARTITION sales_q4 VALUES LESS THAN (1999, 01, 01)
        TABLESPACE tsd );
```

Maintaining Range Partitions

The only maintenance operation to perform on partitions created using the range partitioning method is the merging of partitions. You can use the ALTER TABLE...MERGE PARTITIONS command to merge the contents of two adjacent range partitions into one partition. You might want to do this to keep historical data online in larger partitions. For example, you might want to have daily partitions, with the oldest partition rolled up into weekly partitions, which can then be rolled up into monthly partitions, and so on.

See Also: For more details about range partitioning, see *Oracle8i Concepts.*

For more details about CREATE TABLE...PARTITION syntax, see the *Oracle8i SQL Reference.*

Using the Hash Partitioning Method

Hash partitioning controls the physical placement of data across a fixed 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.

The following example shows how to specify all storage attributes for partitions at the table level:

```
CREATE TABLE scubagear(
   id NUMBER,
   name VARCHAR2 (60))
TABLESPACE ocean
STORAGE (INITIAL 19k)
   PARTITION BY HASH (id)
   PARTITIONS 4;
```

You can store hash partitions in specific tablespaces, as shown in the following statement:

```
CREATE TABLE scubagear (...)
   STORAGE (INITIAL 10k)
   PARTITION BY HASH (id) PARTITIONS 16
   STORE IN (h1to4, h4to8, h8to12, h12to16);
```

Or, you can name and store each hash partition in a specific tablespace:

```
CREATE TABLE product(...)
   STORAGE (INITIAL 10k)
   PARTITION BY HASH (id)
    (PARTITION p1 TABLESPACE h1,
     PARTITION p2 TABLESPACE h2);
```

You can also specify partition-level tablespaces for hash-partitioned indexes:

```
CREATE INDEX bcd_type ON scubagear(id) LOCAL
PARTITIONS 4 STORE IN (ix1, ix2);

CREATE INDEX bcd_type ON scubagear(id) LOCAL
(PARTITION p1 TABLESPACE ix1, PARTITION p2 TABLESPACE ix2,
 PARTITION p3 TABLESPACEix3, PARTITION p4 TABLESPACE ix4);
```

Maintaining Hash Partitions

All current range partition maintenance operations are supported for hash partitions, except for the following:

- ALTER TABLE...SPLIT PARTITION
- ALTER TABLE...DROP PARTITION
- ALTER TABLE...MERGE PARTITIONS

Additionally, there are two maintenance operations specifically for partitions created using the has partitioning method:

- Coalescing Hash Partitions
- Adding Hash Partitions

Coalescing Hash Partitions To remove a single hash partition and redistribute the data, use the following statement:

```
ALTER TABLE scubagear COALESCE PARTITION;
```

Note that the partition being coalesced is determined by the hash function. Also, when you coalesce a hash partition and redistribute the data, local indexes are not maintained. You can coalesce the hash partition in parallel.Local index partitions corresponding to partitions that absorbed rows must be rebuilt from existing partitions.

Adding Hash Partitions To add a single hash partition and redistribute the data, use one of the following statements:

```
ALTER TABLE scubagear ADD PARTITION;
ALTER TABLE scubagear
   ADD PARTITION p3 TABLESPACE t3;
```

Local indexes are not maintained when you add a hash partition. You can also add the hash partition in parallel.

See Also: For detailed syntax information about the CREATE TABLE PARTITION...BY HASH and ALTER TABLE statements, see the *Oracle8i SQL Reference.*

For more details about hash partitioning, see *Oracle8i Concepts.*

Using the Composite Partitioning Method

Composite partitioning partitions data using the range method, and within each partition, subpartitions it using the hash method. 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 a composite partition, you specify the following:

- partitioning method (range)
- partitioning key
- partition descriptions (including partition bounds)
- subpartitioning method (hash)
- subpartitioning columns
- number of subpartitions per partition or descriptions of subpartitions

You may also wish to use the STORE IN clause to specify tablespaces across which each table partition's subpartitions will be spread.

The following statement creates a composite-partitioned table:

```
CREATE TABLE scubagear (equipno NUMBER, equipname VARCHAR(32), price NUMBER)
   PARTITION BY RANGE (equipno) SUBPARTITION BY HASH(equipname)
   SUBPARTITIONS 8 STORE IN (ts1, ts3, ts5, ts7)
   (PARTITION p1 VALUES LESS THAN (1000),
    PARTITION p2 VALUES LESS THAN (2000),
    PARTITION p3 VALUES LESS THAN (MAXVALUE));
```

The following statement shows you can specify subpartition names and names of tablespaces in which subpartitions should be placed.

```
CREATE TABLE scubagear (equipno NUMBER, equipname VARCHAR(32), price NUMBER)
   PARTITION BY RANGE (equipno) SUBPARTITION BY HASH (equipname)
   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));
```

Maintaining Composite Partitions

You can perform all range partition maintenance operations on a composite partitioned table or index, and modify default attributes for table partitions.

Maintaining Composite Subpartitions

This section describes how to accomplish specific subpartition maintenance operations, including:

- Modifying Subpartitions (tables and indexes)
- Rebuilding Subpartitions (indexes only)

- Renaming Subpartitions (tables and indexes)
- Exchanging Subpartitions (tables only)
- Adding Subpartitions (tables only)
- Coalescing Subpartitions (tables only)
- Moving Subpartitions (tables only)
- Truncating Subpartitions (tables only)

Modifying Subpartitions You can mark a subpartition of a local index on a partitioned table marked unusable as follows.

```
ALTER INDEX scuba
   MODIFY SUBPARTITION bcd_types UNUSABLE;
```

You can also allocate or deallocate storage for a subpartition of a table or index using the MODIFY SUBPARTITION clause.

Rebuilding Subpartitions You can rebuild a subpartition to regenerate the data in an index subpartition. The following statement rebuilds a subpartition of a local index on a table:

```
ALTER INDEX scuba
   REBUILD SUBPARTITION bcd_types
   TABLESPACE tbs23 PARALLEL (DEGREE 2);
```

Note that in this example, the index is rebuilt in a different tablespace.

Renaming Subpartitions You can assign new names to subpartitions of a table or index. The following statement shows how to assign a new name to a subpartition of a local index on a table:

```
ALTER INDEX scuba RENAME SUBPARTITION bcd_types TO bcd_brands;
```

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;
```

You can also assign a new name to a subpartition of a table:

```
ALTER TABLE diving RENAME SUBPARTITION locations_us
   TO us_monterey;
```

Exchanging Subpartitions The following statement shows how to convert a subpartition of a table into a nonpartitioned table:

```
ALTER TABLE diving
   EXCHANGE SUBPARTITION locations_us
   WITH TABLE us_ca INCLUDING INDEXES;
```

Adding Subpartitions The following statement shows how to add a subpartition to a partition of a table. The newly added subpartition is populated with rows rehashed from other subpartitions of the same partition as determined by the hash function:

```
ALTER TABLE diving MODIFY PARTITION locations_us
   ADD SUBPARTITION us_monterey TABLESPACE us1;
```

Coalescing Subpartitions The following statement shows how to distribute contents of a subpartition (selected by the RDBMS) of the specified partition of a table into one or more remaining subpartitions (determined by the hash function) of the same partition, and then destroy the selected subpartition. Basically, this operation is the inverse of the ALTER TABLE MODIFY PARTITION ADD SUBPARTITION statement:

```
ALTER TABLE diving MODIFY PARTITION us_locations
   COALESCE PARTITION;
```

Moving Subpartitions The following statement shows how to move data in a subpartition of a table:

```
ALTER TABLE scuba_gear MOVE SUBPARTITION bcd_types
   TABLESPACE tbs23 PARALLEL (DEGREE 2);
```

Truncating Subpartitions The following statement shows how to truncate data in a subpartition of a table:

```
ALTER TABLE diving
   TRUNCATE SUBPARTITION us_locations
      DROP STORAGE;
```

See Also: For more details about the syntax of statements in this section, see the *Oracle8i SQL Reference.*

Creating Partitions

Creating a partitioned table is very similar to creating a table or index: you must use the CREATE TABLE statement with the PARTITION by clause. Also, you must specify the tablespace name for each partition.

The following example shows a CREATE TABLE statement that contains four partitions, one for each quarter's worth of sales. A row with SALE_YEAR=1998, SALE_MONTH=7, and SALE_DAY=18 has the partitioning key (1998, 7, 18), and is in the third partition, in the tablespace TSC. A row with SALE_YEAR=1998, SALE_MONTH=7, and SALE_DAY=1 has the partitioning key (1998, 7, 1), and also is in the third partition.

```
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 ( 1998, 04, 01 )
      TABLESPACE tsa,
    PARTITION sales_q2 VALUES LESS THAN ( 1998, 07, 01 )
      TABLESPACE tsb,
    PARTITION sales_q3 VALUES LESS THAN ( 1998, 10, 01 )
      TABLESPACE tsc,
    PARTITION sales q4 VALUES LESS THAN ( 1999, 01, 01 )
      TABLESPACE tsd);
```

See Also: For more information about the CREATE TABLE statement and PARTITION clause, see *Oracle8i SQL Reference.*

For information about partition keys, partition names, bounds, and equipartitioned tables and indexes, see *Oracle8i Concepts.*

Maintaining Partitions

This section describes how to perform the following specific partition maintenance operations:

- Moving Partitions
- Adding Partitions
- Dropping Partitions
- Coalescing Partitions

- Modifying Partition Default Attributes
- Truncating Partitions
- Splitting Partitions
- Merging Partitions
- Exchanging Table Partitions
- Rebuilding Index Partitions
- Moving the Time Window in a Historical Table
- Quiescing Applications During a Multi-Step Maintenance Operation

See Also: For information about the SQL syntax for DDL statements, see *Oracle8i SQL Reference*.

For information about the catalog views that describe partitioned tables and indexes, and the partitions of a partitioned table or index, see *Oracle8i Reference*.

For information about Import, Export and partitions, see *Oracle8i Utilities*.

For general information about partitioning, see *Oracle8i Concepts*.

Moving Partitions

You can 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

Typically, you can change the physical storage attributes of a partition in a single step via a ALTER TABLE/INDEX...MODIFY PARTITION statement. However, there are some physical attributes, such as TABLESPACE, that you cannot modify via MODIFY PARTITION. In these cases you can use the MOVE PARTITION clause.

Moving Table Partitions

You can use the MOVE PARTITION clause to move a partition. For example, a DBA wishes to move the most active partition to a tablespace that resides on its own disk (in order to balance I/O). The DBA can issue the following statement:

```
ALTER TABLE parts MOVE PARTITION depot2
      TABLESPACE ts094 NOLOGGING;
```

This statement always drops the partition's old segment and creates a new segment, even if you don't specify a new tablespace.

When the partition you are moving contains data, MOVE PARTITION marks the matching partition in each local index, and all global index partitions as unusable. You must rebuild these index partitions after issuing MOVE PARTITION.

Moving Index Partitions

Some operations, such as MOVE PARTITION and DROP TABLE PARTITION, mark 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.

Adding Partitions

This section describes how to add new partitions to a partitioned table and how partitions are added to local indexes.

Adding Table Partitions

You can use the ALTER TABLE...ADD PARTITION statement to add a new partition to the "high" end (the point after the last existing partition). If you wish to add a partition at the beginning or in the middle of a table, or if the partition bound on the highest partition is MAXVALUE, you should instead use the SPLIT PARTITION statement.

When the partition bound of the highest partition is anything other than MAXVALUE, you can add a partition using the ALTER TABLE...ADD PARTITION statement.

For example, a DBA has a table, SALES, which contains data for the current month in addition to the previous 12 months. On January 1, 1999, the DBA adds a partition for January:

```
ALTER TABLE sales
      ADD PARTITION jan96 VALUES LESS THAN ( '01-FEB-1999' )
      TABLESPACE tsx;
```

When there are local indexes defined on the table and you issue the ALTER TABLE...ADD PARTITION statement, a matching partition is also added to each local index. Since Oracle assigns names and default physical storage attributes to

the new index partitions, you may wish to rename or alter them after the ADD operation is complete.

Adding Index Partitions

You cannot explicitly add a partition to a local index. Instead, new partitions are added to local indexes only when you add a partition to the underlying table.

You cannot add a partition to a global index because the highest partition always has a partition bound of MAXVALUE. If you wish to add a new highest partition, use the ALTER INDEX...SPLIT PARTITION statement.

Dropping Partitions

This section describes how to use the ALTER TABLE DROP PARTITION statement to drop table and index partitions and their data.

Dropping Table Partitions

You can use the ALTER TABLE DROP PARTITION statement to drop table partitions.

If there are local indexes defined for the table, ALTER TABLE DROP PARTITION also drops the matching partition from each local index.

Note: You cannot drop the only partition in a table.

Dropping Table Partitions Containing Data and Global Indexes If, however, the partition contains data and one or more global indexes are defined on the table, use either of the following methods to drop the table partition:

1. Leave the global indexes in place during the ALTER TABLE...DROP PARTITION statement. In this situation DROP PARTITION marks all global index partitions unusable, so you must rebuild them afterwards.

   Note: The ALTER TABLE...DROP PARTITION statement not only marks all global index partitions as unusable, it also renders all nonpartitioned indexes unusable. You cannot rebuild the entire partitioned index in a single statement. If you wish to rebuild a partitioned index, you must write a separate REBUILD statement for each partition in the partitioned index. Here, `sal1` is a nonpartitioned index.

```
ALTER TABLE sales DROP PARTITION dec94;
ALTER INDEX sales_area_ix REBUILD sal1;
```

This method is most appropriate for large tables where the partition being dropped contains a significant percentage of the total data in the table.

2. Issue the DELETE command to delete all rows from the partition before you issue the ALTER TABLE...DROP PARTITION statement. The DELETE command updates the global indexes, and also fires triggers and generates redo and undo logs.

Note: You can substantially reduce the amount of logging by setting the NOLOGGING attribute (using ALTER TABLE...MODIFY PARTITION...NOLOGGING) for the partition before deleting all of its rows.

For example, a DBA wishes to drop the first partition, which has a partition bound of 10000. The DBA issues the following statements:

```
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 Table Partitions 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:

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 dec94;
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.

2. Issue the DELETE command to delete all rows from the partition before you issue the ALTER TABLE...DROP PARTITION statement. The DELETE

command 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.

If a global index partition contains data, dropping the partition causes the next highest partition to be marked unusable. For example, a DBA wishes to drop the index partition P1 and P2 is the next highest partition. The DBA 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.

Coalescing Partitions

You can distribute contents of a partition (selected by the RDBMS) of a table partitioned using the hash method into one or more partitions determined by the hash function, and then destroy the selected partition.

The following statement reduces by one the number of partitions in a table by coalescing its last partition:

```
ALTER TABLE ouu1
   COALESCE PARTITION;
```

Modifying Partition Default Attributes

You can modify default attributes of a partition of a local index on tables created using the composite method (or a partition of a composite table).

The following statement changes the (partition-level default) PCTFREE attribute of a partition of a local index on a partitioned table:

```
ALTER INDEX scuba_1
   MODIFY DEFAULT ATTRIBUTES FOR PARTITION bcd_1998
      PCTFREE 25;
```

Truncating Partitions

Use the ALTER TABLE...TRUNCATE PARTITION statement when you wish to remove all rows from a table partition. You cannot truncate an index partition; however, the ALTER TABLE TRUNCATE PARTITION statement truncates the matching partition in each local index.

Truncating Partitioned Tables

You can use the ALTER TABLE...TRUNCATE PARTITION statement to remove all rows from a table partition with or without reclaiming space. If there are local indexes defined for this table, ALTER TABLE...TRUNCATE PARTITION also truncates the matching partition from each local index.

Truncating Table Partitions Containing Data and Global Indexes If, however, the partition contains data and global indexes, use either of the following methods to truncate the table partition:

1. Leave the global indexes in place during the ALTER TABLE TRUNCATE PARTITION statement.

   > **Note:** The ALTER TABLE...TRUNCATE PARTITION statement not only marks all global index partitions as unusable, it also renders all nonpartitioned indexes unusable. You cannot rebuild the entire partitioned index in a single statement. If you wish to rebuild a partitioned index, you must write a separate REBUILD statement for each partition in the partitioned index. Here, `sal1` is a nonpartitioned index.

   ```
   ALTER TABLE sales TRUNCATE PARTITION dec94;
   ALTER INDEX sales_area_ix REBUILD sal1;
   ```

   This method is most appropriate for large tables where the partition being truncated contains a significant percentage of the total data in the table.

2. Issue the DELETE command to delete all rows from the partition before you issue the ALTER TABLE...TRUNCATE PARTITION statement. The DELETE command updates the global indexes, and also fires triggers and generates redo and undo log.

   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 Table Partitions 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:

1. Disable the integrity constraints, issue the ALTER TABLE...TRUNCATE PARTITION statement, then re-enable 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.

2. Issue the DELETE command to delete all rows from the partition before you issue the ALTER TABLE...TRUNCATE PARTITION statement. The DELETE command 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...MODIFY 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.

Splitting Partitions

This form of ALTER TABLE/INDEX divides a partition into two partitions. You can use the SPLIT PARTITION clause when a partition becomes too large and causes backup, recovery or maintenance operations to take a long time. You can also use the SPLIT PARTITION clause to redistribute the I/O load; note that you cannot use this clause for hash partitions.

Splitting Table Partitions

You can split a table partition by issuing the ALTER TABLE...SPLIT PARTITION statement. If there are local indexes defined on the table, this statement also splits the matching partition in each local index. Because Oracle assigns system-generated names and default storage attributes to the new index partitions, you may wish to rename or alter these index partitions after splitting them.

If the partition you are splitting contains data, the ALTER TABLE...SPLIT PARTITION statement marks the matching partitions (there are two) in each local index, as well as all global index partitions, as unusable. You must rebuild these index partitions after issuing the ALTER TABLE...SPLIT PARTITION statement.

Splitting a Table Partition: Scenario In this scenario "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, the DBA issues 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 SYS_P00067;
ALTER INDEX JAF1 REBUILD PARTITION SYS_P00068;
ALTER INDEX VET REBUILD PARTITION VET_parta;
ALTER INDEX VET REBUILD PARTITION VET_partb;
```

> **Note:** You must examine the data dictionary to locate the names assigned to the new local index partitions. In this particular scenario, they are SYS_P00067 and SYS_P00068. If you wish, you can rename them. Also, unless JAF1 already contained partitions fee_katy1 and fee_katy2, names assigned to local index partitions produced by this split will match those of corresponding base table partitions.

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.

The following statement splits the global index partition containing data, QUON1:

```
ALTER INDEX quon1 SPLIT
    PARTITION canada AT VALUES LESS THAN ( 100 ) INTO
    PARTITION canada1 ..., PARTITION canada2 ...);
ALTER INDEX quon1 REBUILD PARTITION canada1;
ALTER INDEX quon1 REBUILD PARTITION canada2;
```

You only need to rebuild if the index partition that you split was unusable.

Merging Partitions

You can merge the contents of two adjacent partitions of a range or composite partitioned table into one. The resulting partition inherits the higher upper bound of the two merged partitions.

The following statement merges two adjacent partitions of a range partitioned table:

```
ALTER TABLE diving
   MERGE PARTITIONS bcd1, bcd2 INTO PARTITION bcd1bcd2;
```

Exchanging Table Partitions

You can convert a partition into a nonpartitioned table, and a table into a partition of a partitioned table by exchanging their data (and index) segments. Exchanging table partitions is most useful when you have an application using nonpartitioned tables which you want to convert to partitions of a partitioned table. For example, you may already have partition views that you wish to migrate into partitioned tables.

Converting a Partition View into a Partitioned Table: Scenario

This scenario describes how to convert a partition view (also called "manual partition") into a partitioned table. The partition view is defined as follows:

```
CREATE VIEW accounts
      SELECT * FROM accounts_jan98
      UNION ALL
      SELECT * FROM accounts_feb98
      UNION ALL
      ...
SELECT * FROM accounts_dec98;
```

To Incrementally Migrate the Partition View to a Partitioned Table

1. Initially, only the two most recent partitions, ACCOUNTS_NOV98 and ACCOUNTS_DEC98, will be migrated from the view to the table by creating the partitioned table. Each partition gets a segment of 2 blocks (as a placeholder).

```
CREATE TABLE accounts_new (...)
    TABLESPACE ts_temp STORAGE (INITIAL 2)
    PARTITION BY RANGE (opening_date)
    (PARTITION jan98 VALUES LESS THAN ('01-FEB-1998'),
     ...
    PARTITION dec98 VALUES LESS THAN ('01-FEB-1998'));
```

2. Use the EXCHANGE command to migrate the tables to the corresponding partitions.

```
ALTER TABLE accounts_new
   EXCHANGE PARTITION nov98 WITH TABLE
   accounts_nov98 WITH VALIDATION;

ALTER TABLE accounts_new
   EXCHANGE PARTITION dec98 WITH TABLE
   accounts_dec98 WITH VALIDATION;
```

   So now the placeholder data segments associated with the NOV98 and DEC98 partitions have been exchanged with the data segments associated with the ACCOUNTS_NOV98and ACCOUNTS_DEC98 tables.

3. Redefine the ACCOUNTS view.

```
CREATE OR REPLACE VIEW accounts
      SELECT * FROM accounts_jan98
      UNION ALL
      SELECT * FROM accounts_feb_98
    UNION ALL
    ...
    UNION ALL
      SELECT * FROM accounts_new PARTITION (nov98)
    UNION ALL
      SELECT * FROM accounts_new PARTITION (dec98);
```

4. Drop the ACCOUNTS_NOV98 and ACCOUNTS_DEC98 tables, which own the placeholder segments that were originally attached to the NOV98 and DEC98 partitions.

5. After all the tables in the UNIONALL view are converted into partitions, drop the view and rename the partitioned to the name of the view being dropped.

```
DROP VIEW accounts;
RENAME accounts_new TO accounts;
```

See Also: For more information about the syntax and usage of the statements in this section, see *Oracle8i SQL Reference.*

Rebuilding Index Partitions

Some operations, such as ALTER TABLE...DROP PARTITION, mark all partitions of a global index unusable. You can rebuild global index partitions in two ways:

1. Rebuild each partition by issuing the ALTER INDEX...REBUILD PARTITION statement (you can run the rebuilds concurrently).
2. Drop the index and re-create it.

Note: This method is more efficient because the table is scanned only once.

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, orders. Historical tables can also be *rollup* tables, which contain summary information derived from the base information via operations such as GROUP BY, AVERAGE, or COUNT.

The time interval in a historical table is a rolling window; DBAs periodically delete sets of rows that describe the oldest transaction, and in turn allocate space for sets of rows that describe the most recent transaction. 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.

To Move the Time Window in a Historical Table 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; the 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, non-prefixed 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. Drop and re-create the global indexes.

```
ALTER INDEX ORDER DROP PARTITION ORDER_IX_CUST;
ALTER INDEX REBUILD PARTITION ORDER_IX_CUST;
```

Quiescing Applications During a Multi-Step Maintenance Operation

Ordinarily, Oracle acquires sufficient locks to ensure that no operation (DML, DDL, 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 DBA's responsibility to ensure that applications (or other maintenance operations) do not interfere with the multi-step operation in progress.

For example, there are referential integrity constraints on the table ORDER, and you do not wish to disable them to drop the partition. Instead, you can replace Step 2 from the previous section with the following:

```
DELETE FROM ORDER WHERE ODATE < TO_DATE( 01-NOV-93 );
ALTER TABLE ORDER DROP PARTITION ORDER_9310;
```

You can ensure that no one inserts new rows into ORDER between the DELETE step and the DROP PARTITION steps by revoking access privileges from an APPLICATION role, which is used in all applications. You can also bring down all user-level applications during a well-defined batch window each night or weekend.

14

Managing Tables

This chapter describes the various aspects of managing tables, and includes the following topics:

- Guidelines for Managing Tables
- Creating Tables
- Altering Tables
- Manually Allocating Storage for a Table
- Dropping Tables
- Index-Organized Tables

Before attempting tasks described in this chapter, familiarize yourself with the concepts in Chapter 12, "Guidelines for Managing Schema Objects".

Guidelines for Managing Tables

This section describes guidelines to follow when managing tables, and includes the following topics:

Use these guidelines to make managing tables as easy as possible.

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 setting storage parameters and defining clusters for tables, based on information from the application developer about how the application works and the types of data expected.

Working with your application developer, carefully plan each table so that the following occurs:

- Tables are normalized.
- Each column is of the proper datatype.
- Columns that allow nulls are defined last, to conserve storage space.
- Tables are clustered whenever appropriate, to conserve storage space and optimize performance of SQL statements.

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's data segment.

See Also: For information about specifying PCTFREE and PCTUSED, see "Managing Space in Data Blocks" on page 12-2.

Specify Transaction Entry Parameters

By specifying the INITRANS and MAXTRANS parameters during the creation of each table, you can affect how much space is initially and can ever be allocated for transaction entries in the data blocks of a table's data segment.

See Also: For information about specifying INITRANS and MAXTRANS, see "Setting Storage Parameters" on page 12-7.

Specify the Location of Each Table

If you have the proper privileges and tablespace quota, you can create a new table in any tablespace that is currently online. Therefore, you should specify the TABLESPACE option in a CREATE TABLE statement to identify the tablespace that will store the new table.

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, make sure 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
- decrease the time needed for database administration

The following examples show how incorrect storage locations of schema objects can affect a database:

- If users' objects are created in the SYSTEM tablespace, the performance of Oracle can be reduced, since both data dictionary objects and user objects must contend for the same datafiles.
- If an application's associated tables are arbitrarily stored in various tablespaces, the time necessary to complete administrative operations (such as backup and recovery) for that application's data can be increased.

See Also: For information about specifying tablespaces, see "Assigning Tablespace Quotas to Users" on page 9-3.

Parallelize Table Creation

You can parallelize the creation of tables created with a subquery in the CREATE TABLE command. Because multiple processes work together to create the table, performance of the table creation can improve.

See Also: For more information about parallel table creation, see the *Oracle8i Parallel Server Concepts and Administration* guide.

For information about the CREATE TABLE command, see the *Oracle8i SQL Reference*.

Consider Creating UNRECOVERABLE Tables

When you create an unrecoverable table, the table cannot be recovered from archived logs (because the needed redo log records are not generated for the unrecoverable table creation). Thus, if you cannot afford to lose the table, you should take a backup after the table is created. In some situations, such as for tables that are created for temporary use, this precaution may not be necessary.

You can create an unrecoverable table by specifying UNRECOVERABLE when you create a table with a subquery in the CREATE TABLE...AS SELECT statement. However, rows inserted afterwards are recoverable. In fact, after the statement is completed, all future statements are fully recoverable.

Creating an unrecoverable table 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.

In general, the relative performance improvement is greater for larger unrecoverable tables than for smaller tables. Creating small unrecoverable tables 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.

Estimate Table Size and Set Storage Parameters

Estimating the sizes of tables before creating them is useful for the following reasons:

- You can use the combined estimated size of tables, along with estimates for indexes, rollback segments, 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 table to better manage the disk space that the table will use. When a table is created, you can set appropriate storage parameters and improve I/O performance of applications that use the table.

  For example, assume that you estimate the maximum size of a table before creating it. If you then set the storage parameters when you create the table, fewer extents will be allocated for the table's data segment, and all of the table's data will be stored in a relatively contiguous section of disk space. This decreases the time necessary for disk I/O operations involving this table.

Whether or not you estimate table size before creation, you can explicitly set storage parameters when creating each nonclustered table. (Clustered tables automatically use the storage parameters of the cluster.) 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.

If you explicitly set the storage parameters for the extents of a table's data segment, try to store the table's data in a small number of large extents rather than a large number of small extents.

Plan for Large Tables

There are no limits on the physical size of tables and extents. You can specify the keyword UNLIMITED for MAXEXTENTS, thereby simplifying your planning for large objects, reducing wasted space and fragmentation, and improving space reuse. However, keep in mind that while Oracle allows an unlimited number of extents, when the number of extents in a table grows very large, you may see an impact on performance when performing any operation requiring that table.

Note: You cannot alter data dictionary tables to have MAXEXTENTS greater than the allowed block maximum.

If you have such tables in your database, consider the following recommendations:

Separate the Table from Its Indexes Place indexes in separate tablespaces from other objects, and on separate disks if possible. If you ever need to drop and re-create an index on a very large table (such as when disabling and enabling a constraint, or re-creating the table), indexes isolated into separate tablespaces can often find contiguous space more easily than those in tablespaces that contain other objects.

Allocate Sufficient Temporary Space If applications that access the data in a very large table perform large sorts, ensure that enough space is available for large temporary segments and that users have access to this space (temporary segments always use the default STORAGE settings for their tablespaces).

Table Restrictions

Before creating tables, make sure you are aware of the following restrictions:

- Tables containing new object types cannot be imported into a pre-Oracle8 database
- You cannot move types and extent tables to a different schema when the original data still exists in the database.
- You cannot merge an exported table into a pre-existing table having the same name in a different schema.

Oracle has a limit on the total number of columns that a table (or attributes that an object type) can have (see *Oracle8i SQL Reference* for this limit.) When you create a table that contains user-defined type data, Oracle maps columns of user-defined type to relational columns for storing the user-defined type data. These "hidden" relational columns 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, the total number of columns that Oracle actually creates for the table may be more than those you specify, because Oracle creates hidden columns to store the user-defined type data. The following formulas determine the total number of columns created for a table with user-defined type data:

Number of columns in an object table:

```
num_columns(object_table) =
    num_columns(object_identifier)
  + num_columns(row_type)
  + number of top-level object columns in the object type of table
  + num_columns(object_type)
```

Number of columns in a relational table:

```
num_columns(relational_table) =
    number of scalar columns in the table
  + number of object columns in the table
  + SUM [num_columns(object_type(i))]   i= 1 -> n
  + SUM [num_columns(nested_table(j))]  j= 1 -> m
  + SUM [num_columns(varray(k))]        k= 1 -> p
  + SUM [num_columns(REF(l))]           l= 1 -> q

where in the given relational table
object_type(i) is the ith object type column and
   n is the total number of such object type columns
nested_table(j) is the jth nested_table column and
   m is the total number of such nested table columns
varray(k) is the kth varray column and
   p is the total number of such varray columns,
REF(l) is the lth REF column and
   q is the total number of such REF columns.

num_columns(object identifier) = 1
num_columns(row_type)          = 1
num_columns(REF)               = 1, if REF is unscoped
                               = 1, if the REF is scoped and the object identifier
                                 is system generated and the REF has no
                                 referential constraint
                               = 2, if the REF is scoped and the object identifier
                                 is system generated and the REF has a
                                 referential constraint
                               = 1 + number of columns in the primary key,
                                 if the object identifier is primary key based
num_columns(nested_table)      = 2
num_columns(varray)            = 1
num_columns(object_type)       = number of scalar attributes in the object type
                               + SUM[num_columns(object_type(i))]     i= 1 -> n
                               + SUM[num_columns(nested_table(j))]    j= 1 -> m
                               + SUM[num_columns(varray(k))]          k= 1 -> p
                               + SUM[num_columns(REF(l))]             l= 1 -> q
```

where in the given object type:

```
object_type(i) is an embedded object type attribute and
   n is the total number of such object type attributes,
nested_table(j) is an embedded nested_table attribute and
   m is the total number of such nested table attributes,
varray(k) is an embedded varray attribute and
   p is the total number of such varray attributes,
REF(l) is an embedded REF attribute and
```

```
   q is the total number of such REF attributes.
```

Example 1

```
CREATE TYPE physical_address_type AS OBJECT
        (no CHAR(4), street CHAR(31), city CHAR(5), state CHAR(3));
CREATE TYPE phone_type AS VARRAY(5) OF CHAR(15);
CREATE TYPE electronic_address_type AS OBJECT
        (phones phone_type, fax CHAR(12), email CHAR(31));
CREATE TYPE contact_info_type AS OBJECT
        (physical_address physical_address_type,
         electronic_address electronic_address_type);
CREATE TYPE employee_type AS OBJECT
        (eno NUMBER, ename CHAR(60),
         contact_info contact_info_type);

CREATE TABLE employee_object_table OF employee_type;
```

To calculate number of columns in employee object table, we first need to calculate number of columns required for employee_type:

```
num_columns(physical_address_type) =
     number of scalar attributes = 4
num_columns(phone_type) =
     num_columns(varray) = 1
num_columns(electronic_address_type) =
     number of scalar attributes
   + num_columns(phone_type)
   = 2 + 1 = 3
num_columns(contact_info_type) =
     num_columns(physical_address_type)
   + num_columns(electronic_address_type)
   = 3 + 4 = 7
num_columns(employee_type) =
     number of scalar attributes
   + num_columns(contact_info_type)
   = 2 + 7 = 9

num_columns (employee_object_table) =
     num_columns(object_identifier)
   + num_columns(row_type)
   + number of top level object columns in employee_type
   + num_columns(employee_type)
   = 1 + 1 + 1 + 9 = 12
```

Example 2:

```
CREATE TABLE employee_relational_table (einfo employee_type);

num_columns (employee_relational_table) =
```

```
     number of object columns in table
   + num_columns(employee_type)
   = 1 + 9 = 10
```

Example 3:

```
CREATE TYPE project_type AS OBJECT (pno NUMBER, pname CHAR(30), budget NUMBER);

CREATE TYPE project_set_type AS TABLE OF project_type;

CREATE TABLE department
        (dno NUMBER, dname CHAR(30),
         mgr REF employee_type REFERENCES employee_object_table,
         project_set project_set_type)
NESTED TABLE project_set STORE AS project_set_nt;

num_columns(department) =
     number of scalar columns
   + num_columns(mgr)
   + num_columns(project_set)
   = 2 + 2 + 2  = 6
```

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. When user SCOTT issues the following statement, he creates a nonclustered table named EMP in his schema and stores it in the USERS tablespace:

```
CREATE TABLE        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 dept_fkey REFERENCES dept)
PCTFREE 10
PCTUSED 40
TABLESPACE users
```

```
STORAGE ( INITIAL 50K
                  NEXT 50K
                  MAXEXTENTS 10
                  PCTINCREASE 25 );
```

Notice that integrity constraints are defined on several columns of the table and that several storage settings are explicitly specified for the table.

See Also: For more information about system privileges, see Chapter 24, "Managing User Privileges and Roles". For more information about tablespace quotas, see Chapter 23, "Managing Users and Resources".

Altering Tables

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.

A table in an Oracle database can be altered for the following reasons:

- to add or drop one or more columns to or from the table
- to add or modify an integrity constraint on a table
- to modify an existing column's definition (datatype, length, default value, and NOT NULL integrity constraint)
- to modify data block space usage parameters (PCTFREE, PCTUSED)
- to modify transaction entry settings (INITRANS, MAXTRANS)
- to modify storage parameters (NEXT, PCTINCREASE)
- to enable or disable integrity constraints or triggers associated with the table
- to drop integrity constraints associated with the table

You can increase the length of an existing column. However, you cannot decrease it unless there are no rows in the table. Furthermore, if you are modifying a table to increase the length of a column of datatype CHAR, realize that this may be a time consuming operation and may 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.

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.

When altering the transaction entry settings (INITRANS, MAXTRANS) of a table, note that a new setting for INITRANS applies only to data blocks subsequently allocated for the table, while a new setting for MAXTRANS applies to all blocks (already and subsequently allocated blocks) of a table.

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.

You can alter a table using the SQL command ALTER TABLE. The following statement alters the EMP table:

```
ALTER TABLE emp
      PCTFREE 30
      PCTUSED 60;
```

WARNING: Before altering a table, familiarize yourself with the consequences of doing so.

If a new column is added to a table, the column is initially null. You can add a column with a NOT NULL constraint to a table only if the table does not contain any rows.

If a view or PL/SQL program unit depends on a base table, the alteration of the base table may affect the dependent object.

See Also: See "Managing Object Dependencies" on page 20-23 for information about how Oracle manages dependencies.

Manually Allocating Storage for a Table

Oracle dynamically allocates additional extents for the data segment of a table, as required. However, you might want to allocate an additional extent for a table explicitly. For example, when using the Oracle Parallel Server, an extent of a table can be allocated explicitly for a specific instance.

A new extent can be allocated for a table using the SQL command ALTER TABLE with the ALLOCATE EXTENT option.

See Also: For information about the ALLOCATE EXTENT option, see *Oracle8i Parallel Server Concepts and Administration.*

Dropping Tables

To drop a table, the table must be contained in your schema or you must have the DROP ANY TABLE system privilege.

To drop a table that is no longer needed, use the SQL command DROP TABLE. The following statement drops the EMP table:

```
DROP TABLE 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, include the CASCADE option in the DROP TABLE command, as shown below:

```
DROP TABLE emp CASCADE CONSTRAINTS;
```

WARNING: 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 20-23 for information about how Oracle manages such dependencies.**
- **All synonyms for a dropped table remain, but return an error when used.**
- **All extents allocated for a nonclustered 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.**

Dropping Columns

Oracle enables you to drop columns from rows in a table, thereby cleaning up unused, and potentially space-demanding columns without having to export/import data, and recreate indexes and constraints.

You can drop columns you no longer need or mark columns to be dropped at a future time when there is less demand on your system's resources.

The following statement drops unused columns from table t1:

```
ALTER TABLE t1 DROP UNUSED COLUMNS;
```

Restrictions

The following restrictions apply to drop column operations:

- You cannot drop a column from an object type table.
- You cannot drop columns from nested tables.
- You cannot drop all columns in a table.
- You cannot drop a partitioning key column.
- You cannot drop a column from tables owned by SYS.
- You cannot drop a parent key column.

See Also: For more information about the syntax used for dropping columns from tables, see the *Oracle8i SQL Reference*.

Index-Organized Tables

This section describes aspects of managing index-organized tables, and includes the following topics:

- What Are Index-Organized Tables?
- Creating Index-Organized Tables
- Maintaining 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?

Index-organized tables are tables with data rows grouped according to the primary key. This clustering is achieved using a *B*-tree index*. B*-tree indexes are special types of index trees that differ from regular table B-tree indexes in that they store both the primary key and non-key columns. The attributes of index-organized tables are stored entirely within the physical data structures for the index.

Why Use Index-Organized Tables?

Index-organized tables provide fast key-based access to table data for queries involving exact match and range searches. Changes to the table data (such as adding new rows, updating rows, or deleting rows) result only in updating the index structure (because there is no separate table storage area).

Also, storage requirements are reduced because key columns are not duplicated in the table and index. The remaining non-key columns are stored in the index structure.

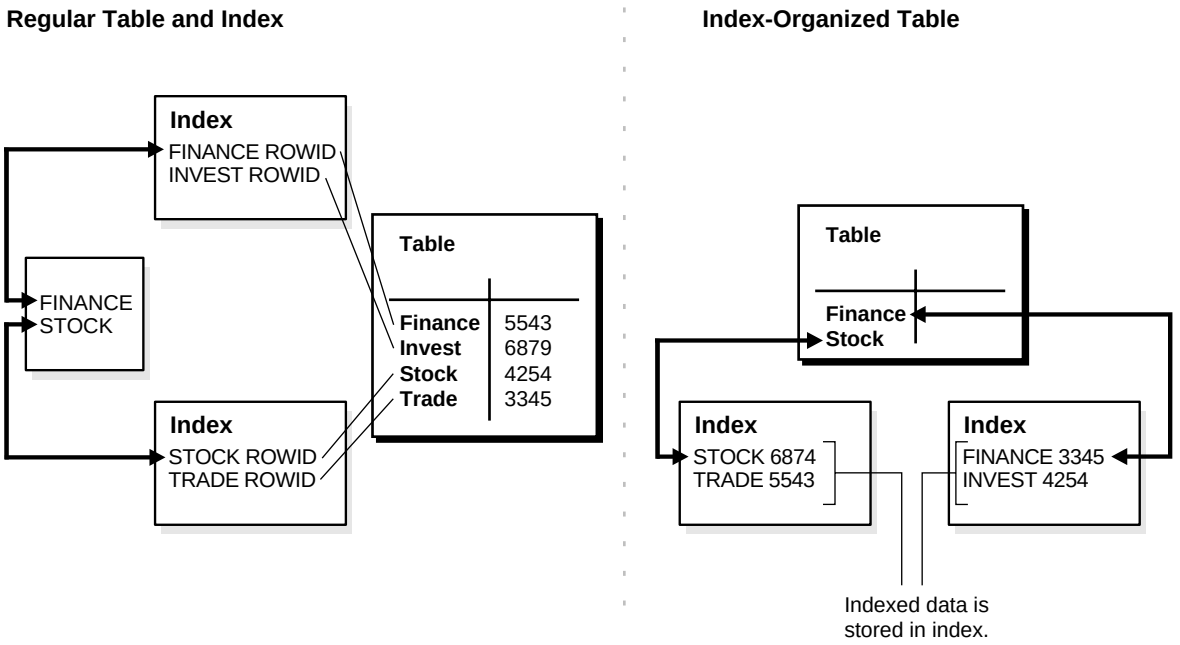
Index-organized tables are particularly useful when you are using applications that must retrieve data based on a primary key. Also, index-organized tables are suitable for modeling application-specific index structures. For example, content-based information retrieval applications containing text, image and audio data require inverted indexes that can be effectively modeled using index-organized tables.

See Also: For more details about index-organized tables, see *Oracle8i Concepts.*

Differences Between Index-Organized and Regular Tables

Index-organized tables are like regular tables with a primary key index on one or more of its columns. However, instead of maintaining two separate storage spaces for the table and B*tree index, an index-organized table only maintains a single B*tree index containing the primary key of the table and other column values.

Figure 14–1 Structure of Regular Table versus Index-Organized Table



Index-organized tables are suitable for accessing data by way of primary key or any key that is a valid prefix of the primary key. Also, there is no duplication of key values because a separate index structure containing the key values and ROWID is not created. Table 14–1 summarizes the difference between an index-organized table and a regular table.

Table 14–1 Comparison of Index-Organized Table with a Regular Table

Regular Table	Index-Organized Table
ROWID uniquely identifies a row; primary key can be optionally specified	Primary key uniquely identifies a row; primary key must be specified
ROWID pseudo-column refers to physical block address	ROWID pseudocolumn refers to primary key-based logical ROWID
Secondary indexes store physical data	Secondary indexes store primary key-based logical ROWID
Can be stored in a hash or index cluster	Cannot be stored in a hash or index cluster

Creating Index-Organized Tables

You can use the CREATE TABLE statement to create index-organized tables; when doing so, you need to provide the following 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). A primary key must be specified for index-organized tables.
- An optional row overflow specification clause, which preserves dense clustering of the B*tree index by storing the row column values exceeding the specified threshold in a separate overflow data segment.

  The *row overflow tablespace* is defined as a percentage of the block size. If a row size is greater than the specified threshold value (PCTTHRESHOLD), the non-key column values are stored in the overflow tablespace. 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 non-key column values that fit the specified threshold, and a pointer to the rest of the row.
- The following example shows the information to provide when creating index-organized tables:

```
CREATE TABLE docindex(
  token char(20),
  doc_id NUMBER,
  token_frequency NUMBER,
  token_offsets VARCHAR2(512),
  CONSTRAINT pk_docindex PRIMARY KEY (token, doc_id))
ORGANIZATION INDEX TABLESPACE ind_tbs
  PCTTHRESHOLD 20
OVERFLOW TABLESPACE ovf_tbs;
```

This example shows that the ORGANIZATION INDEX qualifier specifies an index-organized table, where the key columns and non-key columns reside in an index defined on columns that designate the primary key `(token,doc_id)` for the table.

Index-organized tables can store object types. For example, you can create an index-organized table containing a column of object type `mytype` (for the purpose of this example) as follows:

```
CREATE TABLE iot (c1 NUMBER primary key, c2 mytype)
   ORGANIZATION INDEX;
```

However, you cannot create an index-organized table of object types. For example, the following statement would not be valid:

```
CREATE TABLE iot of mytype ORGANIZATION INDEX;
```

See Also: For more details about the CREATE INDEX statement, see the *Oracle8i SQL Reference.*

Using the AS Subquery

You can create an index-organized table using the AS subquery. Creating an index-organized table in this manner enables you to load the table in parallel by using the PARALLEL option.

The following statement creates an index-organized table (in parallel) by selecting rows from a conventional table, `rt`:

```
CREATE TABLE iot(i primary key, j) ORGANIZATION INDEX PARALLEL (DEGREE 2)
   AS SELECT * FROM rt;
```

See Also: For details about the syntax for creating index-organized tables, see the *Oracle8i SQL Reference.*

Using the Overflow Clause

The overflow clause specified in the earlier example indicates that any non-key columns of rows exceeding 20% of the block size are placed in a data segment stored in the TEXT_COLLECTION_OVERFLOWtablespace . The key columns should fit the specified threshold.

If an update of a non-key column causes the row to decrease in size, Oracle identifies the row piece (head or tail) to which the update is applicable and rewrites that piece.

If an update of a non-key column causes the row to increase in size, Oracle identifies the piece (head or tail) to which the update is applicable and rewrites that row piece. If the update's target turns out to be the head piece, note that this piece may again be broken into 2 to keep the row size below the specified threshold.

The non-key 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 non-key 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: For more information about the ANALYZE statement see the *Oracle8i SQL Reference.*

Using the INCLUDING clause In addition to specifying PCTTHRESHOLD, you can use the INCLUDING <column_name> clause to control which non-key columns are stored with the key columns. Oracle accommodates all non-key columns up to the column specified in the INCLUDING clause in the index leaf block, provided it does not exceed the specified threshold. All non-key columns beyond the column specified in the INCLUDING clause are stored in the overflow area.

For example, you can modify the previous example where an index-organized table was created so that it always has the `token_offsets` column value stored in the overflow area:

```
CREATE TABLE docindex(
  token CHAR(20),
   doc_id NUMBER,
   token_frequency NUMBER,
   token_offsets VARCHAR2(512),
   CONSTRAINT pk_docindex PRIMARY KEY (token, doc_id))
 ORGANIZATION INDEX TABLESPACE ind_tbs
    PCTTHRESHOLD 20
    INCLUDING token_frequency
 OVERFLOW TABLESPACE ovf_tbs;
```

Here, only non-key columns up to `token_frequency` (in this case a single column only) are stored with the key column values in the index leaf block.

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 per 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 iot(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 iot(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 iot(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 A MOVE NOCOMPRESS;
```

See Also: For more details about key compression, see *Oracle8i Concepts* and the *Oracle8i SQL Reference*.

Maintaining Index-Organized Tables

Index-organized tables differ from regular tables only in physical organization; logically, they are manipulated in the same manner. You can use an index-organized table in place of a regular table in INSERT, SELECT, DELETE, and UPDATE statements.

Altering Index-Organized Tables

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 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 `docindex` table as follows:

```
ALTER TABLE docindex PCTTHRESHOLD 15 INCLUDING doc_id;
```

By setting the INCLUDING column to `doc_id`, all the columns that follow `doc_id`, namely, `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, if the `docindex` table did not have an overflow segment, then you can add an overflow segment as follows:

```
ALTER TABLE docindex ADD OVERFLOW TABLESPACE ovf_tbs;
```

See Also: For details about the ALTER TABLE statement, see the *Oracle8i SQL Reference.*

Moving (Rebuilding) Index-Organized Tables

Because index-organized tables are primarily stored in a B*-tree index, you may 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 `docindex` after setting its INITRANS to 10:

```
ALTER TABLE docindex MOVE INITRANS10;
```

You can move index-organized tables with no overflow data segment online using the ONLINE option. For example, if the `docindex` table does not have an overflow data segment, then you can perform the move online as follows:

```
ALTER TABLE docindex MOVE ONLINE INITRANS 10;
```

The following statement rebuilds the index-organized table `docindex` along with its overflow data segment:

```
ALTER TABLE docindex MOVE TABLESPACE ix_tbs OVERFLOW TABLESPACE ov_tbs;
```

And in this last statement, index-organized table `iot` is moved while the LOB index and data segment for C2 are rebuilt:

```
ALTER TABLE iot MOVE LOB (C2) STORE AS (TABLESPACE lob_ts);
```

See Also: For more information about the MOVE option, see the *Oracle8i SQL Reference*.

Scenario: Updating the Key Column

A key column update is logically equivalent to deleting the row with the old key value and inserting the row with the new key value at the appropriate place to maintain the primary key order.

Logically, in the following example, the employee row for `dept_id=20` and `e_id=10` are deleted and the employee row for `dept_id=23` and `e_id=10` are inserted:

```
UPDATE employees
   SET dept_id=23
   WHERE dept_id=20 and e_id=10;
```

Analyzing Index-Organized Tables

Just like conventional tables, index-organized tables are analyzed using the ANALYZE statement:

```
ANALYZE TABLE docindex COMPUTE STATISTICS;
```

The ANALYZE statement 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 queries 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's physical statistics for the table `docindex` as follows:

  ```
  SELECT * FROM DBA_INDEXES WHERE INDEX_NAME= 'PK_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 `docindex` table as follows:

```
SELECT * FROM DBA_TABLES WHERE IOT_TYPE='IOT_OVERFLOW' and IOT_NAME= 'DOCINDEX'
```

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.

For example, you create the following table:

```
CREATE TABLE employees (dept_id INTEGER, e_id INTEGER, e_name
   VARCHAR2, PRIMARY KEY (dept_id, e_id)) ORGANIZATION INDEX;
```

The following queries avoid sorting overhead because the data is already sorted on the primary key:

```
SELECT * FROM employees ORDER BY (dept_id, e_id);
SELECT * FROM employees ORDER BY (dept_id);
```

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 employees ORDER BY (e_id);
SELECT * FROM employees ORDER BY (e_name);
```

Converting Index-Organized Tables to Regular Tables

You can convert index-organized tables to regular tables using the Oracle IMPORT/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: For more details about using IMPORT/EXPORT, see *Oracle8i Utilities.*

15

Managing Views, Sequences and Synonyms

This chapter describes aspects of view management, and includes the following topics:

- Managing Views
- Managing Sequences
- Managing Synonyms

Before attempting tasks described in this chapter, familiarize yourself with the concepts in Chapter 12, "Guidelines for Managing Schema Objects".

Managing Views

A view is a tailored presentation of the data contained in one or more tables (or other views), and takes the output of a query and treats it as a table. You can think of a view as a "stored query" or a "virtual table." You can use views in most places where a table can be used.

This section describes aspects of managing views, and includes the following topics:

- Creating Views
- Modifying a Join View
- Replacing Views
- Dropping Views

Creating Views

To create a view, you must fulfill the requirements listed below:

- 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 may acquire these privileges explicitly or via 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's 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 from it.
- 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 SQL command CREATE VIEW. Each view is defined by a query that references tables, snapshots, or other views. The query that defines a view cannot contain the FOR UPDATE clause. For example, 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 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 is rolled back and returns an error because it attempts to insert a row for department number 30, which could not be selected using the SALES_STAFF view:

```
INSERT INTO sales_staff VALUES (7591, 'WILLIAMS', 30);
```

The following statement creates a 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;
```

The DIVISION1_STAFF view joins information from the EMP and DEPT tables. The CHECK OPTION is not specified in the CREATE VIEW statement for this view.

Expansion of Defining Queries at View Creation Time

In accordance with the ANSI/ISO standard, Oracle expands any wildcard in a top-level view query into a column list when a view is created and stores the resulting query 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;
```

Oracle 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, Oracle can create the view even if the defining query of the view cannot be executed; the view is considered "created with errors." For example, when a view is created that refers to a non-existent 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 option of the CREATE VIEW command:

```
CREATE FORCE VIEW AS ....;
```

By default, views are not created with errors. When a view is created with errors, Oracle returns a message indicating the view was created with errors. The status of a view created with errors is INVALID. If conditions later change so that the query of an invalid view can be executed, the view can be recompiled and become valid (usable).

See Also: For information changing conditions and their impact on views, see "Managing Object Dependencies" on page 20-23.

Modifying a Join View

A *modifiable join view* is a view that contains more than one table in the top-level FROM clause of the SELECT statement, and that does *not* contain any of the following:

- DISTINCT operator
- aggregate functions: AVG, COUNT, GLB, MAX, MIN, STDDEV, SUM, or VARIANCE
- set operations: UNION, UNION ALL, INTERSECT, MINUS
- GROUP BY or HAVING clauses
- START WITH or CONNECT BY clauses
- ROWNUM pseudocolumn

With some restrictions, you can modify views that involve joins. If a view is a join on other nested views, then the other nested views must be mergeable into the top level view.

The examples in following sections use the EMP and DEPT tables. These examples work only if you explicitly define the primary and foreign keys in these tables, or define unique indexes. Following are 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 above, and create a UNIQUE INDEX on DEPT (DEPTNO) to make the following examples work.

See Also: For more information about mergeable views see *Oracle8i Tuning*.

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 and not of the data in the table. For example, if in the EMP table there was at most one employee in each department, then DEPT.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_VIEW, 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

Any UPDATE, INSERT, or DELETE statement on a join view can modify only one underlying base table.

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 ORA-01779 error (''cannot modify a column which maps to a non key-preserved table''), because it attempts to modify the underlying DEPT table, and the DEPT table is not key preserved in the EMP_DEPT view.

In general, all modifiable 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 of repeated tables 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.

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.

In the following view, a DELETE operation cannot 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.

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 like 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.

```
INSERT INTO emp_dept (ename, empno, deptno)
   VALUES ('KURODA', 9010, 77);
```

The following INSERT statement would fail with an ORA-01776 error (''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.

Using the UPDATABLE_ COLUMNS Views

The views described in Table 15–1 can assist you when modifying join views.

Table 15–1 UPDATABLE_COLUMNS Views

View Name	Description
USER_UPDATABLE_COLUMNS	Shows all columns in all tables and views in the user's schema that are modifiable.
DBA_UPDATABLE_COLUMNS	Shows all columns in all tables and views in the DBA schema that are modifiable.
ALL_UPDATABLE_VIEWS	Shows all columns in all tables and views that are modifiable.

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 alter the definition of a view. You can replace views in the following ways:

- You can drop and re-create the view.

> **WARNING: 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 re-granted.**

- You can redefine the view with a CREATE VIEW statement that contains the OR REPLACE option. The OR REPLACE option replaces the current definition of a view and preserves the current security authorizations. For example, assume that you create the SALES_STAFF view as given in the previous example, and grant 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's 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 20-23 for more information on how Oracle manages such dependencies.

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 SQL command DROP VIEW. For example, the following statement drops a view named SALES_STAFF:

```
DROP VIEW sales_staff;
```

Managing Sequences

This section describes various aspects of managing sequences, and includes the following topics:

- Creating Sequences
- Altering Sequences
- Initialization Parameters Affecting Sequences

- Dropping 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 SQL command CREATE SEQUENCE. 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;
```

The CACHE option pre-allocates 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, Oracle reads another set of numbers into the cache.

Oracle 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. Oracle might also skip cached sequence numbers after an export and import; see *Oracle8i Utilities* for details.

See Also: For information about how the Oracle Parallel Server affects cached sequence numbers, see *Oracle8i Parallel Server Concepts and Administration*.

For performance information on caching sequence numbers, see *Oracle8i Tuning*.

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's starting number. To change the starting point of a sequence, drop the sequence and then re-create it. When you perform DDL on sequence numbers you will lose the cache values.

Alter a sequence using the SQL command ALTER SEQUENCE. For example, the following statement alters the EMP_SEQUENCE:

```
ALTER SEQUENCE emp_sequence
    INCREMENT BY 10
    MAXVALUE 10000
    CYCLE
    CACHE 20;
```

Initialization Parameters Affecting Sequences

The initialization parameter SEQUENCE_CACHE_ENTRIES sets the number of sequences that may be cached at any time. If auditing is enabled for your system, allow one additional sequence for the sequence to identify audit session numbers.

If the value for SEQUENCE_CACHE_ENTRIES is too low, Oracle might skip sequence values, as in the following scenario: assume you are using five cached sequences, the cache is full, and SEQUENCE_CACHE_ENTRIES = 4. If four sequences are currently cached, then a fifth sequence replaces the least recently used sequence in the cache and all remaining values (up to the last sequence number cached) in the displaced sequence are lost.

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 SQL command DROP SEQUENCE. 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.

Managing Synonyms

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.

This section includes the following synonym management information:

- Creating Synonyms
- Dropping 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 SQL command CREATE SYNONYM. For example, 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;
```

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 the SQL command DROP SYNONYM. 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).

See Also: For more information about how dropping synonyms can affect other schema objects, see "Managing Object Dependencies" on page 20-23.

16

Managing Indexes

This chapter describes various aspects of index management, and includes the following topics:

- Guidelines for Managing Indexes
- Creating Indexes
- Altering Indexes
- Monitoring Space Use of Indexes
- Dropping Indexes

Before attempting tasks described in this chapter, familiarize yourself with the concepts in Chapter 12, "Guidelines for Managing Schema Objects".

Guidelines for Managing Indexes

This section describes guidelines to follow when managing indexes, and includes the following topics:

An *index* is an optional structure associated with tables and clusters, which you can create explicitly to speed SQL statement execution on a table. Just as the index in this manual helps you locate information faster than if there were no index, an Oracle index provides a faster access path to table data.

The absence or presence of an index does not require a change in the wording of any SQL statement. An index merely offers a fast access path to the data; it affects only the speed of execution. Given a data value that has been indexed, the index points directly to the location of the rows containing that value.

Indexes are logically and physically independent of the data in the associated table. You can create or drop an index any time without affecting the base tables or other indexes. If you drop an index, all applications continue to work; however, access to previously indexed data might be slower. Indexes, being independent structures, require storage space.

Oracle automatically maintains and uses indexes after they are created. Oracle automatically reflects changes to data, such as adding new rows, updating rows, or deleting rows, in all relevant indexes with no additional action by users.

See Also: For information about performance implications of index creation, see *Oracle8i Tuning*.

For more information about indexes, see *Oracle8i Concepts*.

If you have the Oracle Objects option, you can define domain-specific operators and indexing schemes and integrate them into the server. For more information see the *Oracle8i Data Cartridge Developer's Guide*.

Create Indexes After Inserting Table Data

You should create an index for a table after inserting or loading data (via SQL*Loader or Import) into the table. It is more efficient to insert rows of data into a table that has no indexes and then create the indexes for subsequent access. If you create indexes before table data is loaded, every index must be updated every time a row is inserted into the table. You must also create the index for a cluster before inserting any data into the cluster.

When an index is created on a table that already has data, Oracle must use sort space. Oracle uses the sort space in memory allocated for the creator of the index (the amount per user is determined by the initialization parameter SORT_AREA_SIZE), but must also swap sort information to and from temporary segments allocated on behalf of the index creation.

If the index is extremely large, you may want to perform the following tasks.

To Manage a Large Index

1. Create a new temporary segment tablespace.
2. Alter the index creator's temporary segment tablespace.
3. Create the index.
4. Remove the temporary segment tablespace and re-specify the creator's temporary segment tablespace, if desired.

See Also: Under certain conditions, data can be loaded into a table with SQL*Loader's "direct path load" and an index can be created as data is loaded; see *Oracle8i Utilities* for more information.

Limit the Number of Indexes per 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 may be preferable.

Specify Transaction Entry Parameters

By specifying the INITRANS and MAXTRANS parameters during the creation of each index, you can affect how much space is initially and can ever be allocated for transaction entries in the data blocks of an index's segment. You should also leave room for updates and later identify long-term (for example, the life of the index) values for these settings.

See Also: For more information about setting these parameters, see "Setting Storage Parameters" on page 12-7.

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.

See Also: PCTUSED cannot be specified for indexes. See "Managing Space in Data Blocks" on page 12-2 for information about the PCTFREE parameter.

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, then database maintenance may be more convenient (such as tablespace or file backup and application availability or update) and all the related data will always be 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, due to reduced disk contention.

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.

Parallelize Index Creation

You can parallelize index creation. Because multiple processes work together to create the index, Oracle 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: For more information on parallel index creation, see *Oracle8i Tuning*.

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 LOGGING are not archived, you should 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 affect 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.

Estimate Index Size and Set Storage Parameters

Estimating the size of an index before creating one is useful for the following reasons:

- You can use the combined estimated size of indexes, along with estimates for tables, rollback segments, 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 index to better manage the disk space that the index will use. 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 a index before creating it. If you then set the storage parameters when you create the index, fewer extents will be allocated for the table's data segment, and all of the index's data will be 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. As with tables, you can explicitly set storage parameters when creating an index.

See Also: For specific information about storage parameters, see "Setting Storage Parameters" on page 12-7.

Coalescing Indexes

When you encounter index fragmentation (due to improper sizing or increased growth), you can rebuild or coalesce the index. Before you perform either task, though, weigh the costs and benefits of each option and choose the one that works best for your situation. Table 16–1 describes costs and benefits associated with rebuilding and coalescing indexes.

Table 16–1 To Rebuild or Coalesce...That Is the Question.

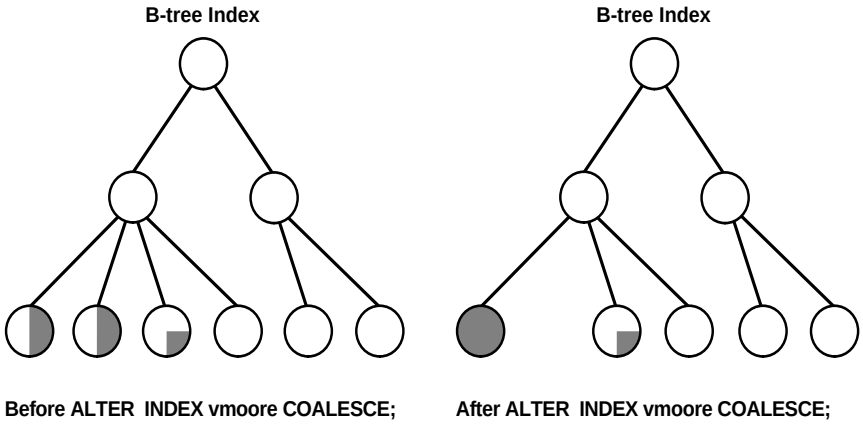
REBUILD	COALESCE
Quickly moves index to another tablespace.	Cannot move index to another tablespace.
Higher costs. Requires more disk space.	Lower costs. Does not require more disk space.
Creates new tree, shrinks height if applicable.	Coalesces leaf blocks within same branch of tree.
Enables you to 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 16–1 illustrates the effect of an ALTER INDEX COALESCE on the index VMOORE. Before performing the operation, the first two leaf blocks are 50% full, which 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 16–1 Coalescing Indexes



Considerations 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 may save time by leaving the constraint enabled rather than dropping and re-creating the large index.

Creating Indexes

This section describes how to create an index, and includes the following topics:

- Creating an Index Associated with a Constraint
- Creating an Index Explicitly
- Creating a Function-Based Index
- Re-creating an Existing Index
- Creating a Key-Compressed Index

To enable a UNIQUE key or PRIMARY KEY (which creates an associated index), the owner of the table needs a quota for the tablespace intended to contain the index, or the UNLIMITED TABLESPACE system privilege.

LOBS, LONG and LONG RAW columns cannot be indexed.

Oracle enforces a UNIQUE key or PRIMARY KEY integrity constraint by creating a unique index on the unique key or primary key. This index is automatically created by Oracle when the constraint is enabled; no action is required by the issuer of the CREATE TABLE or ALTER TABLE statement to create the index. This includes both when a constraint is defined and enabled, and when a defined but disabled constraint is enabled.

In general, it is better to create constraints to enforce uniqueness than it is to use the CREATE UNIQUE INDEX syntax. A constraint's associated index always assumes the name of the constraint; you cannot specify a specific name for a constraint index.

If you do not specify storage options (such as INITIAL and NEXT) for an index, the default storage options of the host tablespace are automatically used.

Creating an Index Associated with a Constraint

You can set the storage options for the indexes associated with UNIQUE key and PRIMARY KEY constraints using the ENABLE clause with the USING INDEX option. The following statement defines a PRIMARY KEY constraint and specifies the associated index's storage option:

```
CREATE TABLE emp (
     empno NUMBER(5) PRIMARY KEY, . . . )
     ENABLE PRIMARY KEY USING INDEX
     TABLESPACE users
     PCTFREE 0;
```

Creating an Index Explicitly

You can create indexes explicitly (outside of integrity constraints) using the SQL command 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 are explicitly specified for the index.

Creating an Index Online

Previously, when creating an index on a table there has always been a DML S-lock on that table during the index build operation, which meant you could not perform DML operations on the base table during the build.

Now, with the ever-increasing size of tables and necessity for continuous operations, you can create and rebuild indexes online—meaning you can update base tables at the same time you are building or rebuilding indexes on that table. Note, though, that there are still DML SS-locks, which means you cannot perform other DDL operations during an online index build.

The following statements perform online index build operations:

```
ALTER INDEX emp_name REBUILD ONLINE;

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. For example, if you wish 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: For more information about the syntax for creating online indexes, see the *Oracle8i SQL Reference*.

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 pre-computed and stored in the index.

Advantages of function-based indexing include support for linguistic sorts based on linguistic sort keys (collation), efficient linguistic collation of SQL statements, and an efficient mechanism for evaluating predicates involving functions.

The following statement defines an index on `Area(geo)`:

```
CREATE INDEX area_index ON rivers (Area(geo)) DESC;
```

The area_index is defined on the function area(geo):

```
SELECT     id, geo Area(geo), desc
FROM       rivers r
WHERE      Area(geo) >5000;
```

In this SQL statement, when Area(geo) is referenced in the WHERE clause, the optimizer considers using the index area_index.

> **Note:** Table owners should have EXECUTE privileges on the functions used in function-based indexes. Also, because a function-based index depends upon any function it is using, it can be invalidated when a function changes.

See Also: For more conceptual information about function-based indexes, see *Oracle8i Concepts.*

For information about function-based indexing and application development, see the *Oracle8i Application Developer's Guide - Fundamentals.*

Example 1

The following statement creates a function-based index, idx on table emp:

```
CREATE INDEX idx ON emp (UPPER(emp_name));
```

Now the SELECT statement uses the function-based index on UPPER(emp_name) to retrieve all employees with names like :KEYCOL:

```
SELECT * FROM emp WHERE UPPER(emp_name) like :KEYCOL;
```

SELECT statements can use either an index range scan (the expression is a prefix of the index) or index full scan (preferable when the index specifies a high degree of parallelism).

```
CREATE INDEX idx ON t (a + b * (c - 1), a, b);
SELECT a FROM t WHERE a + b * (c - 1) < 100;
```

Example 2

You can also use function-based indexes to support NLS sort index as well. NLSSORT is a function that returns a sort key that has been given a string. Thus, if you want to build an index on name using NLSSORT, issue the following statement:

```
CREATE INDEX nls_index ON t_table (NLSSORT(name, 'NLS_SORT = German'));
```

This statement creates the `nls_index` on table `t_table` with the collation sequence German.

Now, to select from NLS_SORT:

```
SELECT * FROM t_table ORDER BY name;
```

Rows will be ordered using the collation sequence in German.

Example 3

Another use for function-based indexing is to perform non-case-sensitive searches:

```
CREATE INDEX case_insensitive_idx ON emp_table (UPPER(empname));
```

A query on this new index would look like the following:

```
SELECT * FROM emp_table WHERE UPPER(empname) = 'JOE';
```

Example 4

This example also illustrates the most common uses of function-based indexing: a case-insensitive sort and language sort.

```
CREATE INDEX empi ON emp
   UPPER ((ename), NLSSORT(ename));
```

Here, an NLS_SORT specification does not appear in the NLSSORT argument because NLSSORT looks at the session setting for the language of the linguistic sort key. If you wish to use a language other than the language specified for the session setting, replace NLSSORT(ename) in the example above with the following:

```
NLSSORT(ename, NLS_SORT='German')
```

This line directs the sort to use a German linguistic sort key.

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 will be marked invalid. You must use the ANALYZE VALIDATE INDEX statement to validate this index.

See Also: For more information about function-based indexing, see *Oracle8i Concepts* and *Oracle8i SQL Reference.*

Re-creating an Existing Index

Before re-creating or rebuilding an existing index, compare the costs and benefits associated with rebuilding to those associated with coalescing indexes as described in Table 16–1 on page 16-6.

You can create an index using an existing index as the data source. Creating an index in this manner allows you to change storage characteristics or move to a new tablespace. Re-creating an index based on an existing data source also removes intra-block fragmentation. In fact, compared to dropping the index and using the CREATE INDEX command, re-creating an existing index offers better performance.

Issue the following statement to re-create an existing index:

```
ALTER INDEX index_name REBUILD;
```

The REBUILD clause must immediately follow the index name, and precede any other options. Also, the REBUILD clause cannot be used in conjunction with the DEALLOCATE UNUSED clause.

See Also: For more information on the ALTER INDEX command and optional clauses, see the *Oracle8i SQL Reference.*

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 per index block while improving performance.

Key compression can be useful in the following situations:

- You have a non-unique index where ROWID is appended to make the key unique. If you use key compression here, the duplicate key will be stored as a prefix entry on the index block without the ROWID. The remaining rows will be suffix entries consisting of only the ROWID
- You have a unique multi-column index.

You can enable key compression using the COMPRESS clause.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. For example, the following statement will compress away duplicate occurrences of a key in the index leaf block.

```
CREATE INDEX  emp_ename (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: For more details about the CREATE INDEX statement, see the *Oracle8i SQL Reference.*

Altering Indexes

To alter an index, your schema must contain the index or you must have the ALTER ANY INDEX system privilege. You can alter an index only to change the transaction entry parameters or to change the storage parameters; you cannot change its column structure.

Alter the storage parameters of any index, including those created by Oracle to enforce primary and unique key integrity constraints, using the SQL command ALTER INDEX. For example, the following statement alters the EMP_ENAME index:

```
ALTER INDEX emp_ename
      INITRANS 5
      MAXTRANS 10
```

```
    STORAGE (PCTINCREASE 50);
```

When you alter the transaction entry settings (INITRANS, MAXTRANS) of an index, a new setting for INITRANS applies only to data blocks subsequently allocated, while a new setting for MAXTRANS applies to all blocks (currently and subsequently allocated blocks) of an index.

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 also adjust storage parameters by issuing an ALTER TABLE statement that includes the ENABLE clause with the USING INDEX option. For example, the following statement changes the storage options of the index defined in the previous section:

```
ALTER TABLE emp
     ENABLE PRIMARY KEY USING INDEX
     PCTFREE 5;
```

Monitoring Space Use of Indexes

If key values in an index are inserted, updated, and deleted frequently, the index may or may not use its acquired space efficiently over time. Monitor an index's efficiency of space usage at regular intervals by first analyzing the index's structure and then querying the INDEX_STATS view:

```
SELECT pct_used FROM sys.index_stats WHERE name = 'indexname';
```

The percentage of an index's space usage will vary according to how often index keys are inserted, updated, or deleted. Develop a history of an index's average efficiency of space usage by performing the following sequence of operations several times:

- analyzing statistics
- validating the index
- checking PCT_USED
- dropping and re-creating (or coalescing) the index

When you find that an index's space usage drops below its average, you can condense the index's space by dropping the index and rebuilding it, or coalescing it.

See Also: For information about analyzing an index's structure, see "Analyzing Tables, Indexes, and Clusters" on page 20-3.

Dropping Indexes

To drop an index, the index must be contained in your schema, or you must have the DROP ANY INDEX system privilege.

You might want to drop an index for any of the following reasons:

- 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's 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.

> **Note:** If a table is dropped, all associated indexes are dropped automatically.

You cannot drop only the index associated with an enabled UNIQUE key or PRIMARY KEY constraint. To drop a constraint's associated index, you must disable or drop the constraint itself.

```
DROP INDEX emp_ename;
```

See Also: For information about analyzing indexes, see "Analyzing Tables, Indexes, and Clusters" on page 20-3.

For more information about dropping a constraint's associated index, see "Managing Integrity Constraints" on page 20-13.

17

Managing Clusters

This chapter describes aspects of managing clusters (including clustered tables and indexes), and includes the following topics:

- Guidelines for Managing Clusters
- Creating Clusters
- Altering Clusters
- Dropping Clusters

Before attempting tasks described in this chapter, familiarize yourself with the concepts in Chapter 12, "Guidelines for Managing Schema Objects".

Guidelines for Managing 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, which 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 physically stores all rows for each department from both the EMP and DEPT tables in the same data blocks. You should not use clusters for tables that are frequently accessed individually.

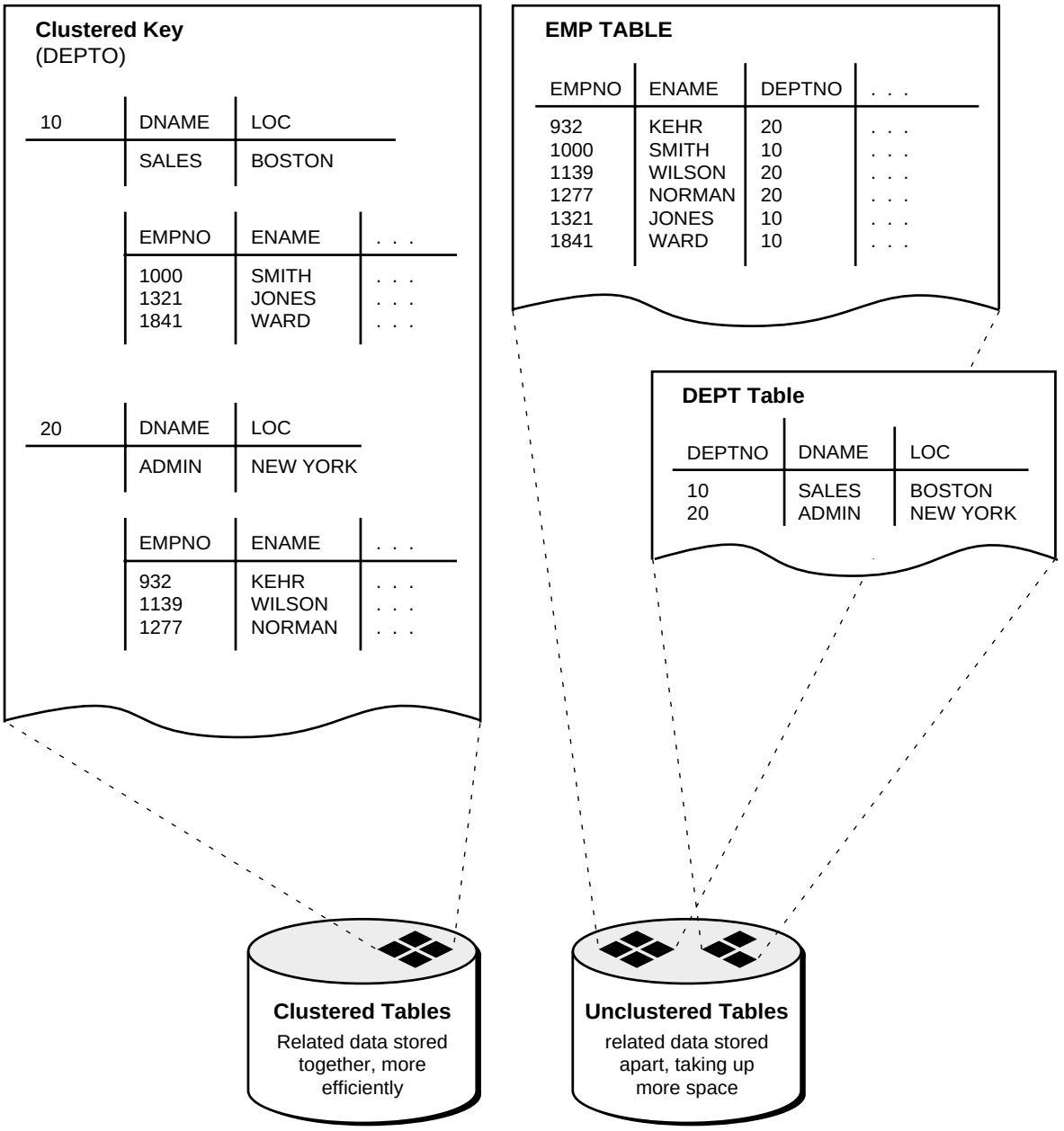
Because clusters store related rows of different tables together in the same data blocks, properly used clusters offer two primary benefits:

- 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 non-clustered table format. For example, 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.

Figure 17–1 Clustered Table Data



The following sections describe guidelines to consider when managing clusters, and includes 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: For more information about clusters, see *Oracle8i Concepts*.

Choose Appropriate Tables for the Cluster

Use clusters to store one or more tables that are primarily queried (not predominantly inserted into or updated) and for which the queries often join data of multiple tables in the cluster or retrieve related data from a single table.

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.

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 per 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 below).

Too many rows per 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.

See Also: For information about characteristics of a good index, see "Guidelines for Managing Indexes" on page 16-2.

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's data segment. Note that PCTFREE and PCTUSED parameters set for tables created in a cluster are ignored; clustered tables automatically use the settings set for the cluster.

See Also: For more information about setting PCTFREE and PCTUSED, see "Managing Space in Data Blocks" on page 12-2.

Specify the Space Required by an Average Cluster Key and Its Associated Rows

The CREATE CLUSTER command has an optional argument, SIZE, which is the estimated number of bytes required by an average cluster key and its associated rows. Oracle 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, Oracle stores only one cluster key and its associated rows in each data block of the cluster's data segment. Although block size can vary from one operating system to the next, the rule of one key per 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 option 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

Following are benefits of estimating a cluster's size before creating it:

- You can use the combined estimated size of clusters, along with estimates for indexes, rollback segments, 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 non-clustered 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

This section describes how to create clusters, and includes the following topics:

- Creating Clustered Tables
- Creating Cluster Indexes

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 can create a cluster using the SQL 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);
```

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 can create a table in a cluster using the SQL CREATE TABLE statement with the CLUSTER option. The EMP and DEPT tables can be created in the EMP_DEPT cluster using the following statements:

```
CREATE TABLE dept (
   deptno NUMBER(3) PRIMARY KEY, . . . )
   CLUSTER emp_dept (deptno);

CREATE TABLE emp (
   empno NUMBER(5) PRIMARY KEY,
   ename VARCHAR2(15) NOT NULL,
   . . .
   deptno NUMBER(3) REFERENCES dept)
   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 don't have to match, but their structure does.

Creating Cluster Indexes

To create a cluster index, one of the following conditions must be true:

- Your schema contains the cluster and you have the CREATE INDEX system privilege.
- 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
   INITRANS 2
   MAXTRANS 5
   TABLESPACE users
   STORAGE (INITIAL 50K
      NEXT 50K
      MINEXTENTS 2
      MAXEXTENTS 10
      PCTINCREASE 33)
   PCTFREE 5;
```

Several storage settings are explicitly specified for the cluster and cluster index.

See Also: See Chapter 24, "Managing User Privileges and Roles" for more information about system privileges, and Chapter 23, "Managing Users and Resources" for information about tablespace quotas.

Altering Clusters

You can alter an existing cluster to change the following settings:

- data block space usage parameters (PCTFREE, PCTUSED)
- the average cluster key size (SIZE)
- transaction entry settings (INITRANS, MAXTRANS)
- storage parameters (NEXT, PCTINCREASE)

To alter a cluster, your schema must contain the cluster or you must have the ALTER ANY CLUSTER system privilege.

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 settings (INITRANS, MAXTRANS) of a cluster, a new setting for INITRANS applies only to data blocks subsequently

allocated for the cluster, while a new setting for MAXTRANS applies to all blocks (already and subsequently allocated blocks) of a 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;
```

Altering Cluster Tables and Cluster Indexes

You can alter clustered tables using the SQL 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"). Oracle uses the parameters of the cluster for all clustered tables. Therefore, you can use the ALTER TABLE command 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.

> **Note:** When estimating the size of cluster indexes, remember that the index is on each cluster key, not the actual rows; therefore, each key will only appear once in the index.

Manually Allocating Storage for a Cluster

Oracle dynamically allocates additional extents for the data segment of a cluster as required. In some circumstances, however, you might want to allocate an additional extent for a cluster explicitly. For example, when using the Oracle Parallel Server, 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 option.

See Also: For information about altering tables, see "Altering Tables" on page 14-10.

You alter cluster indexes exactly as you do other indexes. For more information, see "Altering Indexes" on page 16-13.

For more information about the CLUSTER parameter in the ALTER CLUSTER statement, see *Oracle8i Parallel Server Concepts and Administration*.

Dropping Clusters

This section describes aspects of dropping clusters, and includes the following topics:

- Dropping Clustered Tables
- Dropping Cluster Indexes

A cluster can be dropped if the tables within the cluster are no longer necessary. When a cluster is dropped, so are the tables within the cluster and the corresponding cluster index; all extents belonging to both the cluster's 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.

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 have to have 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 table's cluster, other clustered tables, or the cluster index. A clustered table is dropped just as a non-clustered table is dropped—with the DROP TABLE statement.

Note: When you drop a single table from a cluster, Oracle 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 option. Drop an individual table from a cluster (using the DROP TABLE statement) only if you want the rest of the cluster to remain.

See Also: For information about dropping a table, see "Dropping Tables" on page 14-12.

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.

To drop a cluster that contains no tables, and its cluster index, use the SQL 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 option of the DROP CLUSTER statement, as follows:

```
DROP CLUSTER emp_dept INCLUDING TABLES;
```

If the INCLUDING TABLES option 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 option of the DROP CLUSTER statement, as shown in the following example:

```
DROP CLUSTER emp_dept INCLUDING TABLES CASCADE CONSTRAINTS;
```

Oracle returns an error if you do not use the CASCADE CONSTRAINTS option and constraints exist.

See Also: For information about dropping an index, see "Dropping Indexes" on page 16-15.

18

Managing Hash Clusters

This chapter describes how to manage hash clusters, and includes the following topics:

- Guidelines for Managing Hash Clusters
- Altering Hash Clusters
- Dropping Hash Clusters

See Also: Before attempting tasks described in this chapter, familiarize yourself with the concepts in Chapter 12, "Guidelines for Managing Schema Objects".

Guidelines for Managing Hash Clusters

This section describes guidelines to consider before attempting to manage hash clusters, and includes the following topics:

- Advantages of Hashing
- Disadvantages of Hashing
- Estimate Size Required by Hash Clusters and Set Storage Parameters

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 non-clustered table with an index or an index cluster. With an indexed table or index cluster, Oracle locates the rows in a table using key values that Oracle stores in a separate index. To use hashing, you create a hash cluster and load tables into it. Oracle physically stores the rows of a table in a hash cluster and retrieves them according to the results of a hash function.

Oracle uses a *hash function* to generate a distribution of numeric values, called *hash values*, which 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, Oracle applies the hash function to the row's cluster key value; the resulting hash value corresponds to a data block in the cluster, which Oracle 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, Oracle 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: For more information about hash clusters, see *Oracle8i Concepts*.

Advantages of Hashing

If you opt to use indexing rather than hashing, consider whether to store a table individually or as part of a cluster.

Hashing is most advantageous 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.

Disadvantages of Hashing

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 like 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 and continually growing. If a table grows without limit, the space required over the life of the table (its cluster) cannot be pre-determined.

- 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 pre-allocate the space that the hash cluster will eventually need.

See Also: For more information about creating hash clusters and specifying hash functions see the *Oracle8i SQL Reference.*

For information about hash functions and specifying user-defined hash functions, see *Oracle8i Concepts.*

Even if you decide to use hashing, a table can still have separate indexes on any columns, including the cluster key. See the *Oracle8i Application Developer's Guide - Fundamentals* for additional recommendations.

Estimate Size Required by Hash Clusters and Set Storage Parameters

As with index clusters, it is important to estimate the storage required for the data in a hash cluster.

Oracle 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 per block is approximately 1900 bytes (data block size minus overhead), and that the STORAGE and HASH parameters are specified in the CREATE CLUSTER command 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 per 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).

Creating Hash Clusters

After creating a hash cluster, you can create tables in the cluster. A hash cluster is created using the SQL command CREATE CLUSTER. The following statement creates a cluster named TRIAL_CLUSTER that stores the TRIAL table, clustered by the TRIALNO column:

```
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);
```

The following sections explain setting the parameters of the CREATE CLUSTER command specific to hash clusters.

See Also: For additional information about creating tables in a cluster, guidelines for setting other parameters of the CREATE CLUSTER command, and the privileges required to create a hash cluster, see "Creating Clusters" on page 17-6.

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;
```

Oracle rounds the HASHKEY value up to the nearest prime number, so this cluster has a maximum of 503 hash key values, each of size 512 bytes:

> **Note:** The single table option is valid only for hash clusters. HASHKEYS must also be specified.

See Also: For more information about the CREATE CLUSTER statement, see the *Oracle8i SQL Reference*.

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.Iif 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 Oracle's internal hash function.

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 the above conditions apply, you can distribute rows in the cluster so that each unique cluster key value hashes, with no collisions, to a unique hash value. If these conditions do not apply, omit this option 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 per 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.
- If the hash cluster does not use the internal hash function (if you specified HASH IS) and you expect little or no collisions, you can set SIZE as estimated; 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 increase SIZE according to Table 18–1.

Table 18–1 SIZE Increase Chart

Available Space per Block/Calculated SIZE	Setting for SIZE
1	Calculated SIZE
2	Calculated SIZE + 15%
3	Calculated SIZE + 12%
4	Calculated SIZE + 8%
>4	Calculated 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 above adjustments and use the estimated value for SIZE.

Setting HASHKEYS

For maximum distribution of rows in a hash cluster, Oracle rounds the HASHKEYS value up to the nearest prime number.

Controlling Space in Hash Clusters: Examples

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).

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. Since 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 per data block. HASHKEYS can be set to the number of rows in the table, 10000, rounded up to the next highest prime number, 10007:

```
CREATE CLUSTER emp_cluster (empno
NUMBER)
. . .
SIZE 55
HASH IS empno HASHKEYS 10007;
```

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 per department. Note that 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 pre-estimated SIZE (the average amount of space required to hold all rows per department) is 55 bytes * 10, or 550 bytes. Using this value for SIZE, only three hash keys can be assigned per 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 (see previous section about setting SIZE for clarification), only three hash keys are assigned per data block, leaving more space for rows from expected collisions.

HASHKEYS can be set to the number of unique department numbers, 1000, rounded up to the next highest prime number, 1009:

```
CREATE CLUSTER emp_cluster (deptno NUMBER)
. . .
SIZE 620
HASH IS deptno HASHKEYS 1009;
```

Altering Hash Clusters

You can alter a hash cluster with the SQL ALTER CLUSTER statement:

```
ALTER CLUSTER emp_dept . . . ;
```

The implications for altering a hash cluster are identical to those for altering an index cluster. However, note that the SIZE, HASHKEYS, and HASH IS parameters cannot be specified in an ALTER CLUSTER statement. You must re-create the cluster to change these parameters and then copy the data from the original cluster.

See Also: For more information about altering an index cluster, see ""Altering Clusters" on page 17-8.

Dropping Hash Clusters

You can drop a hash cluster using the SQL DROP CLUSTER statement:

```
DROP CLUSTER emp_dept;
```

A table in a hash cluster is dropped using the SQL DROP TABLE statement. The implications of dropping hash clusters and tables in hash clusters are the same for index clusters.

See Also: For more information about dropping clusters, see "Dropping Clusters" on page 17-10.

19

Detecting and Repairing Data Block Corruption

Oracle provides different methods for detecting and correcting data block corruption. One method 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. DBMS_REPAIR uses the following approach to address corruptions:

- Step 1: Detect and Report Corruptions
- Step 2: Evaluate the Costs and Benefits of Using DBMS_REPAIR
- Step 3: Make Objects Usable
- Step 4: Repair Corruptions and Rebuild Lost Data

> **Note:** Any corruption that involves the loss of data requires analysis and understanding of how that data fits into the overall database system. Hence, 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 may lose data and logical inconsistencies can be introduced; therefore you need to weigh the gains and losses associated with using DBMS_REPAIR.

DBMS_REPAIR Package Contents

Table 19–1 describes the procedures that make up the DBMS_REPAIR package.

Table 19–1 DBMS_REPAIR Procedures

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 corrupt.
`dump_orphan_keys`	Reports index entries that point to rows in corrupt data blocks.
`rebuild_freelists`	Rebuilds an object's freelists.
`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 DBMS_REPAIR repair and orphan key tables. **Note:** These tables are always created in the SYS schema.

Step 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 19–2 describes the different detection methodologies.

Table 19–2 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.

Table 19–2 Comparison of Corruption Detection Methods

Detection Method	Description
DB_BLOCK_CHECKING	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 respectively.

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: For more information about DB_VERIFY, see *Oracle8i Utilities.*

ANALYZE: Corruption Reporting

The ANALYZE TABLE...VALIDATE STRUCTURE statement validates the structure of the analyzed object. If Oracle successfully validates the structure, a message confirming its validation is returned to you. If Oracle encounters corruption in the structure of the object, an error message is returned to you. In this case, you would drop and re-create the object.

See Also: For more information about the ANALYZE statement, see the *Oracle8i SQL Reference.*

DB_BLOCK_CHECKING (Block Checking Initialization Parameter)

You can set block checking for instances via the DB_BLOCK_CHECKING parameter (the default value is TRUE); this checks data and index blocks whenever

they are modified. DB_BLOCK_CHECKING is a dynamic parameter, modifiable by the ALTER SYSTEM SET statement.

Step 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.

A first step is to answer the following questions:

1. 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.

2. What other options are available for addressing block corruptions?

   Assuming the data is available from another source, drop, re-create and re-populate the object. Another option is to issue the CREATE TABLE...AS SELECT statement from the corrupt table to create a new one.

   You can ignore the corruption by excluding corrupt rows from select statements.

   Perform media recovery.

3. What logical corruptions or side effects will be introduced when you use DBMS_REPAIR to make an object usable? Can these be addressed? What is the effort required to do so?

   You may not have access to rows in blocks marked corrupt. However, a block may be marked corrupt even though there are still rows that you can validly access.

   Referential integrity constraints may be broken when blocks are marked corrupt. If this occurs, disable and re-enable the constraint; any inconsistencies will be reported. After fixing all issues, you should be able to successfully re-enable the constraint.

   Logical corruption may occur when there are triggers defined on the table. For example, if rows are re-inserted, should insert triggers be fired or not? You can address these issues only if you understand triggers and their use in your installation.

   Freelist blocks may be inaccessible. If a corrupt block is at the head or tail of a freelist, space management reinitializes the freelist. There then may be blocks

that should be on a freelist, that aren't. You can address this by running the `rebuild_freelists` procedure.

Indexes and tables may be 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.

4. 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` procedures 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.

Step 3: Make Objects Usable

In this step 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's 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 will skip 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 may 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 will 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 giving different results.

Step 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 Freelists Using the rebuild_freelists Procedure

When a block marked "corrupt" is found at the head or tail of a freelist, the freelist is reinitialized and an error is returned. Although this takes the offending block off the freelist, it causes you to lose freelist access to all blocks that followed the corrupt block.

You can use the `rebuild_freelists` procedure to reinitialize the freelists. The object is scanned, and if it is appropriate for a block to be on the freelist, it is added to the master freelist. Freelist groups are handled by meting out the blocks in an equitable fashion—a block at a time. Any blocks marked "corrupt" in the object are ignored during the rebuild.

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_freelist` 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.

DBMS_REPAIR Procedures

This sections contains detailed descriptions of the DBMS_REPAIR procedures.

check_object

The check_object procedure checks the specified objects, and populates the repair table with information about corruptions and repair directives. Validation consists of block checking all blocks in the object. You may optionally specify a range, partition name, or subpartition name when you wish to check a portion of an object.

```
procedure check_object(schema_name IN varchar2,
   object_name IN varchar2,
   partition_name IN varchar2 DEFAULT NULL,
   object_type IN binary_integer  DEFAULT  TABLE_OBJECT,
   repair_table_name IN varchar2 DEFAULT 'REPAIR_TABLE',
   flags  IN binary_integer  DEFAULT NULL,
   relative_fno IN binary_integer  DEFAULT NULL,
   block_start IN binary_integer DEFAULT NULL,
   block_end IN binary_integer DEFAULT NULL,
   corrupt_count OUT binary_integer)
```

Table 19–3 The check_object Procedure

Argument	Description
schema_name	Schema name of the object to be checked.
object_name	Name of the table or index to be checked.
partition_name (optional)	Partition or subpartition name to be checked. If this is a partitioned object, and partition_name is not specified, then all partitions and subpartitions are checked. If this is a partitioned object, and the specified partition contains subpartitions, then all subpartitions are checked.
object_type (optional)	Type of the object to be processed. Must be either TABLE_OBJECT or INDEX_OBJECT. The default is TABLE_OBJECT.
repair_table_name (optional)	Name of the repair table to be populated. The table must exist in the SYS schema. Use the admin_tables procedure to create a repair table. The default name is 'REPAIR_TABLE'.
flags (optional)	Reserved for future use.
relative_fno (optional)	Relative file number. Used when specifying a block range.
block_start (optional)	The first block to process if specifying a block range. May be specified only if the object is a single table, partition, or subpartition.
block_end (optional)	The last block to process if specifying a block range. May be specified only if the object is a single table, partition, or subpartition. If only one of block_start or block_end is specified, then the other defaults to the first or last block in the file respectively.
corrupt_count	The number of corruptions reported.

fix_corrupt_blocks

Use this 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 effected, the associated row in the repair table is updated with a fix timestamp.

```
procedure fix_corrupt_blocks(
   schema_name IN varchar2,
   object_name IN varchar2,
   partition_name IN varchar2 DEFAULT NULL,
   object_type IN binary_integer DEFAULT TABLE_OBJECT,
   repair_table_name IN varchar2 DEFAULT 'REPAIR_TABLE',
   flags  IN boolean DEFAULT NULL,
   fix_count OUT binary_integer)
```

Table 19–4 The fix_corrupt_blocks Procedure

Argument	Description
schema_name	Schema name.
object_name	Name of the object with corrupt blocks to be fixed.
partition_name (optional)	Partition or subpartition name to be processed. If this is a partitioned object, and partition_name is not specified, then all partitions and subpartitions are processed. If this is a partitioned object, and the specified partition contains subpartitions, then all subpartitions are processed.
object_type (optional)	Type of the object to be processed. Must be either TABLE_OBJECT or INDEX_OBJECT. The default is TABLE_OBJECT.
repair_table_name (optional)	Name of the repair table with the repair directives. Must exist in the SYS schema.
flags (optional)	Reserved for future use.
fix_count	The number of blocks fixed.

dump_orphan_keys

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 table.

If the repair table is specified, then any corrupt blocks associated with the base table are handled in addition to all data blocks that are marked software corrupt. Otherwise, only blocks that are marked corrupt are handled.

This information may be useful for rebuilding lost rows in the table and for diagnostic purposes.

```
procedure dump_orphan_keys(
   schema_name IN varchar2,
   object_name IN varchar2,
```

```
   partition_name IN varchar2 DEFAULT NULL,
   object_type IN binary_integer DEFAULT INDEX_OBJECT,
   repair_table_name IN varchar2 DEFAULT  'REPAIR_TABLE',
   orphan_table_name IN varchar2 DEFAULT 'ORPHAN_KEY_TABLE',
   key_count OUT binary_integer)
```

Table 19–5 The dump_orphan_keys Procedure

Argument	Description
schema_name	Schema name.
object_name	Object name.
partition_name (optional)	Partition or subpartition name to be processed. If this is a partitioned object, and partition_name is not specified, then all partitions and subpartitions are processed. If this is a partitioned object, and the specified partition contains subpartitions, then all subpartitions are processed.
object_type (optional)	Type of the object to be processed. The default is INDEX_OBJECT.
repair_table_name (optional)	Name of the repair table that has information regarding corrupt blocks in the base table. The specified table must exist in the SYS schema. The admin_tables procedure is used to create the table.
orphan_table_name (optional)	Name of the orphan key table to populate with information regarding each index entry that refers to a row in a corrupt data block. The specified table must exist in the SYS schema. The admin_tables procedure is used to create the table.
key_count	Number of index entries processed.

rebuild_freelists

Rebuilds the freelists for the specified object. All free blocks are placed on the master freelist. All other freelists are zeroed. If the object has multiple freelist groups, then the free blocks are distributed among all freelists, allocating to the different groups in round-robin fashion.

```
procedure rebuild_freelists(
   schema_name IN varchar2,
   object_name IN varchar2,
   partition_name IN varchar2 DEFAULT  NULL,
```

```
   object_type IN binary_integer DEFAULT TABLE_OBJECT);
```

Table 19–6 The rebuild_freelists Procedure

Argument	Description
schema_name	Schema name.
object_name	Name of the object whose freelists are to be rebuilt.
partition_name (optional)	Partition or subpartition name whose freelists are to be rebuilt. If this is a partitioned object, and partition_name is not specified, then all partitions and subpartitions are processed. If this is a partitioned object, and the specified partition contains subpartitions, then all subpartitions are processed.
object_type (optional)	Type of the object to be processed. Must be either TABLE_OBJECT or INDEX_OBJECT. The default is TABLE_OBJECT.

skip_corrupt_blocks

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.

```
procedure skip_corrupt_blocks(
   schema_name IN varchar2,
   object_name IN varchar2,
   partition_name IN varchar2 DEFAULT NULL,
   object_type IN binary_integer DEFAULT TABLE_OBJECT,
   flags IN boolean DEFAULT SKIP_FLAG);
```

Table 19–7 The skip_corrupt_blocks Procedure

Argument	Description
schema_name	Schema name of the object to be processed.
object_name	Name of the object.
partition_name (optional)	Partition or subpartition name to be processed. If this is a partitioned object, and partition_name is not specified, then all partitions and subpartitions are processed. If this is a partitioned object, and the specified partition contains subpartitions, then all subpartitions are processed.
object_type (optional)	Type of the object to be processed. Must be either TABLE_OBJECT or CLUSTER_OBJECT. The default is TABLE_OBJECT.
flags (optional)	If SKIP_FLAG is specified, turns on the skip of software corrupt blocks for the object during index and table scans. If NOSKIP_FLAG is specified, scans that encounter software corrupt blocks return an ORA-1578.

admin_tables

Provides administrative functions for repair and orphan key tables.

```
procedure admin_tables(
   table_name IN varchar2,
   table_type IN binary_integer,
   action IN binary_integer,
   tablespace IN varchar2 DEFAULT NULL);
```

Table 19–8 The admin_tables Procedure

Argument	Description
table_name	Name of the table to be processed. Defaults to 'ORPHAN_KEY_TABLE' or 'REPAIR_TABLE' based on the specified table_type. When specified, the table name must have the appropriate prefix, 'ORPHAN_' or 'REPAIR_'.
table_type	Type of table, must be one of ORPHAN_TABLE or REPAIR_TABLE.
action	Indicates what administrative action to perform. Must be CREATE_ACTION, PURGE_ACTION, or DROP_ACTION. If the table already exists, and CREATE_ACTION is specified, then an error is returned. PURGE_ACTION indicates to delete all rows in the table that are associated with non-existent objects. If the table does not exist, and DROP_ACTION is specified, then an error is returned. When CREATE_ACTION and DROP_ACTION are specified, an associated view named DBA_<table_name> is created and dropped respectively. The view is defined so that rows associated with non-existent objects are eliminated. Created in the SYS schema.
tablespace (optional)	Indicates the tablespace to use when creating a table. By default, SYS's default tablespace is used. An error is returned if the tablespace is specified and the action is not CREATE_ACTION.

DBMS_REPAIR Exceptions

942	repair table doesn't exist
1418	specified index doesn't exist
24120	invalid parameter
24121	can't specify CASCADE_FLAG and a block range
24122	invalid block range
24124	invalid action parameter specified
24126	CASCADE_FLAG specified and object is not a table

24127	tablespace specified and action is not CREATE_ACTION
24128	partition specified for non-partitioned object
24129	invalid orphan key table name - must have 'ORPHAN_' prefix
24129	specified repair table does not start with 'REPAIR_' prefix
24131	repair table has incorrect columns
24132	repair table name is too long

20

General Management of Schema Objects

This chapter describes general schema object management issues that fall outside the scope of Chapters 11 through 19, and includes the following topics:

- Creating Multiple Tables and Views in a Single Operation
- Renaming Schema Objects
- Analyzing Tables, Indexes, and Clusters
- Truncating Tables and Clusters
- Enabling and Disabling Triggers
- Managing Integrity Constraints
- Managing Object Dependencies
- Managing Object Name Resolution
- Changing Storage Parameters for the Data Dictionary
- Displaying Information About Schema Objects

Creating Multiple Tables and Views in a Single Operation

To create schema objects you must have the required privileges for any included operation. For example, to create multiple tables using the CREATE SCHEMA command, you must have the privileges required to create tables.

You can create several tables and views and grant privileges in one operation using the SQL statement CREATE SCHEMA. The CREATE SCHEMA statement is useful if you want to guarantee the creation of several tables and 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. 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 extensions to the ANSI CREATE TABLE and CREATE VIEW commands; this includes the STORAGE clause.

Renaming Schema Objects

To rename an object, you must own it. You can rename schema objects in either of the following ways:

- drop and re-create the object
- rename the object using the SQL statement RENAME

If you drop and re-create an object, all privileges granted for that object are lost. Privileges must be re-granted 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, 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 rename 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: For more information about how Oracle manages object dependencies, see "Managing Object Dependencies" on page 20-23.

Analyzing Tables, Indexes, and Clusters

This section describes how to analyze tables, indexes, and clusters, and includes the following topics:

- Using Statistics for Tables, Indexes, and Clusters
- Validating Tables, Indexes, and Clusters
- Listing Chained Rows of Tables and Clusters

You can analyze a table, index, or cluster to gather data about it, or to verify the validity of its storage format. To analyze a table, cluster, or index, you must own the table, cluster, or index or have the ANALYZE ANY system privilege.

These schema objects can also be analyzed to collect or update statistics about specific objects. When a DML statement is issued, the statistics for the referenced objects are used to determine the most efficient execution plan for the statement. This optimization is called "cost-based optimization." The statistics are stored in the data dictionary.

A table, index, or cluster can be analyzed to *validate* the structure of the object. 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 a schema object is corrupt, you can drop and re-create it.

A table or cluster can be analyzed to collect information about chained rows of the table or cluster. 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.

See Also: For more information about analyzing tables, indexes, and clusters for performance statistics and the optimizer, see *Oracle8i Tuning*.

For information about analyzing index-organized tables, see Chapter 14, "Managing Tables".

Using Statistics for Tables, Indexes, and Clusters

Statistics about the physical storage characteristics of a table, index, or cluster can be gathered and stored in the data dictionary using the SQL statement ANALYZE with the STATISTICS option. Oracle can use these statistics when cost-based optimization is employed to choose the most efficient execution plan for SQL statements accessing analyzed objects. You can also use statistics generated by this command to write efficient SQL statements that access analyzed objects.

You can compute or estimate statistics using the ANALYZE statement, with either the COMPUTE STATISTICS or ESTIMATE STATISTICS option:

COMPUTE STATISTICS	When computing statistics, an entire object is scanned to gather data about the object. This data is used by Oracle to compute exact statistics about the object. Slight variances throughout the object are accounted for in these computed statistics. Because an entire object is scanned to gather information for computed statistics, the larger the size of an object, the more work that is required to gather the necessary information.

ESTIMATE STATISTICS	When estimating statistics, Oracle gathers representative information from portions of an object. This subset of information provides reasonable, estimated statistics about the object. The accuracy of estimated statistics depends upon how representative the sampling used by Oracle is. Only parts of an object are scanned to gather information for estimated statistics, so an object can be analyzed quickly. You can optionally specify the number or percentage of rows that Oracle should use in making the estimate.

Note: When calculating statistics for tables or clusters, the amount of temporary space required to perform the calculation is related to the number of rows specified. For COMPUTE STATISTICS, enough temporary space to hold and sort the entire table plus a small overhead for each row is required. For ESTIMATE STATISTICS, enough temporary space to hold and sort the requested sample of rows plus a small overhead for each row is required. For indexes, no temporary space is required for analyzing.

See Also: For more information about the SQL statement ANALYZE, see the *Oracle8i SQL Reference*.

For more information about the data dictionary views containing statistics, see the *Oracle8i Reference*.

Viewing Object Statistics

Whether statistics for an object are computed or estimated, the statistics are stored in the data dictionary. The statistics can be queried using the following data dictionary views:

- USER_INDEXES, ALL_INDEXES, DBA_INDEXES
- USER_TABLES, ALL_TABLES, DBA_TABLES
- USER_TAB_COLUMNS, ALL_TAB_COLUMNS, DBA_TAB_COLUMNS

Note: Rows in these views contain entries in the statistics columns only for indexes, tables, and clusters for which you have gathered statistics. The entries are updated for an object each time you ANALYZE the object.

Table Statistics You can gather the following statistics on a table:

> **Note:** The * symbol indicates that the numbers will always be an exact value when computing statistics.

- number of rows
- number of blocks that have been used *
- number of blocks never used
- average available free space
- number of chained rows
- average row length
- number of distinct values per column
- the second smallest value per column *
- the second largest value per column *

> **Note:** Statistics for all indexes associated with a table are automatically gathered when the table is analyzed.

Index Statistics You can gather the following statistics on an index:

- index level *
- number of leaf blocks
- number of distinct keys
- average number of leaf blocks / key
- average number of data blocks / key
- clustering factor

Cluster Statistics The only statistic that can be gathered for a cluster is the average cluster key chain length; this statistic can be estimated or computed. Statistics for tables in a cluster and all indexes associated with the cluster's tables (including the

cluster key index) are automatically gathered when the cluster is analyzed for statistics.

> **Note:** If the data dictionary currently contains statistics for the specified object when an ANALYZE statement is issued, the new statistics replace the old statistics in the data dictionary.

Computing Statistics

The following statement computes statistics for the EMP table:

```
ANALYZE TABLE emp COMPUTE STATISTICS;
```

The following query estimates statistics on the EMP table, using the default statistical sample of 1064 rows:

```
ANALYZE TABLE emp ESTIMATE STATISTICS;
```

To specify the statistical sample that Oracle should use, include the SAMPLE option with the ESTIMATE STATISTICS option. You can specify an integer that indicates either a number of rows or index values, or a percentage of the rows or index values in the table. The following statements show examples of each option:

```
ANALYZE TABLE emp
   ESTIMATE STATISTICS
      SAMPLE 2000 ROWS;
ANALYZE TABLE emp
   ESTIMATE STATISTICS
      SAMPLE 33 PERCENT;
```

In either case, if you specify a percentage greater than 50, or a number of rows or index values that is greater than 50% of those in the object, Oracle computes the exact statistics, rather than estimating.

Removing Statistics for a Schema Object

You can remove statistics for a table, index, or cluster from the data dictionary using the ANALYZE command with the DELETE STATISTICS option. For example, you might want to delete statistics for an object if you do not want cost-based optimization to be used for statements regarding the object. The following statement deletes statistics for the EMP table from the data dictionary:

```
ANALYZE TABLE emp DELETE STATISTICS;
```

Shared SQL and Analyzing Statistics

Analyzing a table, cluster, or index can affect current shared SQL statements, which are statements currently in the shared pool. Whenever an object is analyzed to update or delete statistics, all shared SQL statements that reference the analyzed object are flushed from memory so that the next execution of the statement can take advantage of the new statistics.

You can call the following procedures:

DBMS_UTILITY.-ANALYZE_SCHEMA()
This procedure takes two arguments: the name of a schema and an analysis method ('COMPUTE', 'ESTIMATE', or 'DELETE'). It gathers statistics on all of the objects in the schema.

DBMS_DDL.-ANALYZE_OBJECTS()
This procedure takes four arguments: the type of object ('CLUSTER', 'TABLE', or 'INDEX'), the schema of the object, the name of the object, and an analysis method ('COMPUTE', 'ESTIMATE', or 'DELETE'). It gathers statistics on the object.

You should call these procedures periodically to update the statistics.

Validating Tables, Indexes, and Clusters

To verify the integrity of the structure of a table, index, cluster, or snapshot, use the ANALYZE command 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. If a table, index, or cluster is corrupt, you should drop it and re-create it. If a snapshot is corrupt, perform a complete refresh and ensure that you have remedied the problem; if not, drop and re-create the snapshot.

The following statement analyzes the EMP table:

```
ANALYZE TABLE emp VALIDATE STRUCTURE;
```

You can validate an object and all related objects by including the CASCADE option. The following statement validates the EMP table and all associated indexes:

```
ANALYZE TABLE emp VALIDATE STRUCTURE CASCADE;
```

Listing Chained Rows of Tables and Clusters

You can look at the chained and migrated rows of a table or cluster using the ANALYZE command with the LIST CHAINED ROWS option. The results of this command are stored in a specified table created explicitly to accept the information returned by the LIST CHAINED ROWS option.

To create an appropriate table to accept data returned by an ANALYZE...LIST CHAINED ROWS statement, use the UTLCHAIN.SQL script provided with Oracle. The UTLCHAIN.SQL script creates a table named CHAINED_ROWS in the schema of the user submitting the script.

After a CHAINED_ROWS table is created, you can specify it when using 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: The name and location of the UTLCHAIN.SQL script are operating system-dependent; see your operating system-specific Oracle documentation.

For more information about reducing the number of chained and migrated rows in a table or cluster, see *Oracle8i Tuning*.

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, you may have 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 three options:

1. Using the DELETE statement

   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;
   ```

2. Using the DROP and CREATE statements

   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 ( . . . );
   ```

3. Using TRUNCATE

   You can delete all rows of the table using the SQL statement TRUNCATE. For example, the following statement truncates the EMP table:

   ```
   TRUNCATE TABLE emp;
   ```

Using DELETE

If there are many rows present in a table or cluster when using the DELETE command, significant system resources are consumed as the rows are deleted. For example, CPU time, redo log space, and rollback 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 wipe out the entire object.

Using DROP and CREATE

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

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 rollback 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 the user's associated schema. Also, any user that 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.

A hash cluster cannot be truncated. Also, tables within a hash or index cluster cannot 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 command or drop and re-create the table.

The REUSE STORAGE or DROP STORAGE options of the TRUNCATE command 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. Also note that the storage parameters for a truncated table, cluster, or associated indexes are not changed as a result of the truncation.

See Also: See Chapter 25, "Auditing Database Use", for information about auditing.

Enabling and Disabling Triggers

Database *triggers* are procedures that are stored in the database and activated ("fired") when a user makes a particular modification, such as adding a row to a table. You can use triggers to supplement the standard capabilities of Oracle 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 are implicitly executed when any of the following statements are issued against an associated table:

- INSERT
- UPDATE
- DELETE
- STARTUP
- SHUTDOWN
- LOGON

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: For more details about triggers, see *Oracle8i Concepts*.

For details about creating triggers, see *Oracle8i SQL Reference*.

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;
```

Disabling Triggers

You may want to temporarily disable a trigger if one of the following conditions is true:

- An object that the trigger references is not available.
- You have to 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 or statements about data in a database. Enabled constraints check data as it is entered or updated in the database and prevent data that does not conform to the constraint's rule from being entered. Validated constraints can guarantee uniqueness, master-detail relationships, compliance with an expression, or that NULLs are not present.

These rules, or statements are always true when the constraint is enabled and validated. However, the statement may or may not be true when the constraint is disabled (or put in "enable novalidate" state) because data in violation of the integrity constraint can be in the database.The following sections explain the mechanisms and procedures for managing integrity constraints:

- Integrity Constraint States
- Deferring Constraint Checks
- Managing Constraints That Have Associated Indexes
- Setting Integrity Constraints Upon Definition
- Modifying Existing Integrity Constraints
- Dropping Integrity Constraints
- Reporting Constraint Exceptions

See Also: You can identify exceptions to a specific integrity constraint while attempting to enable the constraint. See "Reporting Constraint Exceptions" on page 20-21.

For general information about integrity constraints, see *Oracle8i Concepts.*

Integrity Constraint States

An integrity constraint defined on a table can be in one of four states:

disable novalidate	When a constraint is disable novalidated, the rule defined by the constraint is not enforced on the data values in the columns included in the constraint; however, the definition of the constraint is retained in the data dictionary. This mode is useful when you are performing a data warehouse rollup or load and you want to speed up the load process.
enable novalidate	A table with enable novalidate constraints can contain invalid data, but it is not possible to add new invalid data to it. Useful as an intermediate state before validating the data in the table using enable validate. This ensures no new data can violate the constraint, and no locks are held when taking constraints from enable no validate to enable validate. This mode is useful when you don't want to enable the constraint to check for exceptions, for example, after a data warehouse load.
enable and validate	An enabled constraint is enforced and known to be valid (validity of table data is checked). The definition of the constraint is stored in the data dictionary. This is the normal operational state for constraint processing. This state is useful for preventing invalid data entry during regular OLTP processing.
disable validate	Allows you to have a unique constraint without an index. Tables in this state cannot be updated. Enables you to load nonpartitioned data into a partitioned table using the EXCHANGE PARTITION statement. Also useful when you have tables for data warehousing purposes and want to minimize space usage.

Disabling Constraints

To enforce the rules defined by integrity constraints, the constraints should always be enabled. However, you may wish to temporarily disable 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 bullets above.

Enable Novalidate Constraints

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.

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 examine all rows violating constraints in the EXCEPTIONS table

See Also: For details about the EXCEPTIONS table, see *Oracle8i Reference.*

Integrity Constraint States: Procedures and Benefits

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 permitted

Deferring Constraint Checks

When Oracle 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.

> **Note:** You cannot issue a SET CONSTRAINT statement inside a trigger.

See Also: For more details about the SET CONSTRAINTS statement, see the *Oracle8i SQL Reference.*

For general information about constraints, see *Oracle8i Concepts.*

How To Defer Constraint Checks

Select Appropriate Data You may wish to defer constraint checks on UNIQUE and FOREIGN keys if the data you are working with has any of the following characteristics:

- tables are snapshots

- tables that contain a large amount of data being manipulated by another application, which may or may not return the data in the same order
- update cascade operations on FOREIGN keys

When dealing with bulk data being manipulated by outside applications, you can defer checking constraints for validity until the end of a transaction.

Ensure Constraints Are Created Deferrable After you have identified and selected the appropriate tables, make sure the tables' FOREIGN, UNIQUE and PRIMARY key constraints are created deferrable. You can do so by issuing a statement similar to the following:

```
CREATE TABLE dept (
     deptno NUMBER PRIMARY KEY,
     dname VARCHAR2 (30)
     );
CREATE TABLE emp (
     empno NUMBER,
     ename VARCHAR2 (30),
     deptno NUMBER REFERENCES (dept),
     CONSTRAINT epk PRIMARY KEY (empno) DEFERRABLE,
     CONSTRAINT efk FOREIGN KEY (deptno)
     REFERENCES (dept. deptno) DEFERRABLE);
INSERT INTO dept VALUES (10, 'Accounting');
INSERT INTO dept VALUES (20, 'SALES');
INSERT INTO emp VALUES (1, 'Corleone', 10);
INSERT INTO emp VALUES (2, 'Costanza', 20);
COMMIT;

SET CONSTRAINT efk DEFERRED;
UPDATE dept SET deptno = deptno + 10
     WHERE deptno = 20;

SELECT * from emp ORDER BY deptno;
EMPNO   ENAME          DEPTNO
-----   -------------- -------
   1    Corleone       10
   2    Costanza       20
UPDATE emp SET deptno = deptno + 10
     WHERE deptno = 20;
SELECT * FROM emp ORDER BY deptno;

EMPNO   ENAME          DEPTNO
-----   -------------- -------
   1    Corleone       10
   2    Costanza       30
COMMIT;
```

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 will fail and the constraint causing the error will be identified. If you commit while constraints are violated, the transaction will be rolled back and you will receive an error message.

Managing Constraints That Have Associated Indexes

When you create a UNIQUE or PRIMARY key, Oracle checks to see if an existing index can be used to enforce uniqueness for the constraint. If there is no such index, Oracle creates one.

When Oracle is using a unique index to enforce a constraint, and constraints associated with the unique index are dropped or disabled, the index is dropped.

While enabled foreign keys reference a PRIMARY or UNIQUE key, you cannot disable or drop the PRIMARY or UNIQUE key constraint or the index.

> **Note:** Deferrable UNIQUE and PRIMARY keys all must use non-unique indexes.

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 by including one of the following clauses in the constraint definition:

- ENABLE

- DISABLE
- ENABLE [VALIDATE]
- DISABLE [NOVALIDATE]
- ENABLE NOVALIDATE
- DISABLE VALIDATE

If none of these clauses are identified in a constraint's definition, Oracle 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 of the table that violate the integrity constraint. The definition of the constraint is allowed because its rule is not enforced.

See Also: For more information about constraint exceptions, see "Reporting Constraint Exceptions" on page 20-21.

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 may fail because rows of the table may violate the integrity constraint. In this case, the statement is rolled back and the constraint definition is not stored and not enabled.

To enable a UNIQUE key or PRIMARY KEY, which creates an associated index, the owner of the table also needs a quota for the tablespace intended to contain the index, or the UNLIMITED TABLESPACE system privilege.

Modifying Existing Integrity Constraints

You can use the ALTER TABLE statement to enable, disable or modify a constraint.

Disabling Enabled Constraints

The following statements disable integrity constraints:

```
ALTER TABLE dept
    DISABLE CONSTRAINT dname_ukey;
ALTER TABLE dept
    DISABLE PRIMARY KEY,
    DISABLE UNIQUE (dname, loc);
```

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;
```

Dropping Integrity 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 the DROP clause. The following two statements drop integrity constraints:

```
ALTER TABLE dept
    DROP UNIQUE (dname, loc);
ALTER TABLE emp
    DROP PRIMARY KEY,
    DROP CONSTRAINT dept_fkey;
```

Dropping UNIQUE key and PRIMARY KEY constraints drops the associated unique indexes. Also, 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.

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.

You cannot use the CREATE TABLE statement to determine which rows are in violation. 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.

Note: 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 submitting the script UTLEXCPT.SQL, which creates a table named EXCEPTIONS. You can create additional exceptions tables with different names by modifying and re-submitting 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, the following rows might be placed in the table EXCEPTIONS by the previous statement:

```
SELECT * FROM exceptions;

ROWID                 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 example:

```
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 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, other users may issue statements creating new exceptions. You can avoid this by enable novalidating the constraint before you start eliminating exceptions.

See Also: The exact name and location of the UTLEXCPT.SQL script is operating system specific. For more information, see your operating system-specific Oracle documentation.

Managing Object Dependencies

This section describes the various object dependencies, and includes 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 in 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 column MODIFY column) 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

Table 20–1 Operations that Affect Object Status (Cont.)

Operation	Resulting Status of Object	Resulting Status of Dependent Objects
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 or procedure[2]	VALID if there are no error; INVALID if there are syntax or authorization errors	INVALID
REVOKE object privilege[3] ON objectTO / 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]

[1] May cause dependent objects to be made INVALID, if object did not exist earlier.

[2] Stand-alone 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.

Oracle automatically recompiles an invalid view or PL / SQL program unit the next time it is used. In addition, a user can force Oracle to recompile a view or program unit using the appropriate SQL command with the COMPILE parameter. 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 command with the COMPILE parameter to recompile a view. The following statement recompiles the view EMP_DEPT contained in your schema:

```
ALTER VIEW emp_dept COMPILE;
```

Manually Recompiling Procedures and Functions

To recompile a stand-alone 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 parameter to recompile a stand-alone procedure or function. The following statement recompiles the stored procedure UPDATE_SALARY contained in your schema:

```
ALTER PROCEDURE update_salary COMPILE;
```

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 parameter to recompile either a package body or both a package specification and body. The following statements recompile just the body, and the body and specification of the package ACCT_MGMT, respectively:

```
ALTER PACKAGE acct_mgmt COMPILE BODY;
ALTER PACKAGE acct_mgmt COMPILE PACKAGE;
```

Managing Object Name Resolution

This section describes how Oracle resolves an object name.

1. First. Oracle 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, Oracle 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. If no schema object is found in the current schema, Oracle 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. If no public synonym is found, Oracle 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 a object in the previously qualified schema or there is not a second piece, Oracle returns an error.

   If no schema is found in Step c, the object cannot be qualified and Oracle 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 Oracle resolves the reference locally. For example, it resolves a synonym to a remote table's global object name. The partially resolved statement is shipped to the remote database, and the remote Oracle completes the resolution of the object as described here.

Changing Storage Parameters for the Data Dictionary

This section describes aspects of changing data dictionary storage parameters, and includes the following topics:

- Structures in the Data Dictionary
- Errors that Require Changing Data Dictionary Storage

If your database is very large or contains an unusually large number of objects, columns in tables, constraint definitions, users, or other definitions, the tables that make up the data dictionary might at some point be unable to acquire additional extents. For example, a data dictionary table may need an additional extent, but there is not enough contiguous space in the SYSTEM tablespace. If this happens,

you cannot create new objects, even though the tablespace intended to hold the objects seems to have sufficient space. To remedy this situation, you can change the storage parameters of the underlying data dictionary tables to allow them to be allocated more extents, in the same way that you can change the storage settings for user-created segments. For example, you can adjust the values of NEXT or PCTINCREASE for the data dictionary table.

> **WARNING: Exercise caution when changing the storage settings for the data dictionary objects. If you choose inappropriate settings, you could damage the structure of the data dictionary and be forced to re-create your entire database. For example, if you set PCTINCREASE for the data dictionary table USER$ to 0 and NEXT to 2K, that table will quickly reach the maximum number of extents for a segment, and you will not be able to create any more users or roles without exporting, re-creating, and importing the entire database.**

Structures in the Data Dictionary

The following tables and clusters contain the definitions of all the user-created objects in the database:

SEG$	segments defined in the database (including temporary segments)
OBJ$	user-defined objects in the database (including clustered tables); indexed by I_OBJ1 and I_OBJ2
UNDO$	rollback segments defined in the database; indexed by I_UNDO1
FET$	available free extents not allocated to any segment
UET$	extents allocated to segments
TS$	tablespaces defined in the database
FILE$	files that make up the database; indexed by I_FILE1
FILEXT$	datafiles with the AUTOEXTEND option set on
TAB$	tables defined in the database (includes clustered tables); indexed by I_TAB1

CLU$	clusters defined in the database
IND$	indexes defined in the database; indexed by I_IND1
ICOL$	columns that have indexes defined on them (includes individual entries for each column in a composite index); indexed by I_ICOL1
COL$	columns defined in tables in the database; indexed by I_COL1 and I_COL2
CON$	constraints defined in the database (includes information on constraint owner); indexed by I_CON1 and I_CON2
CDEF$	definitions of constraints in CON$; indexed by I_CDEF1, I_CDEF2, and I_CDEF3
CCOL$	columns that have constraints defined on them (includes individual entries for each column in a composite key); indexed by I_CCOL1
USER$	users and roles defined in the database; indexed by I_USER1
TSQ$	tablespace quotas for users (contains one entry for each tablespace quota defined for each user)
C_OBJ#	cluster containing TAB$, CLU$, ICOL$, IND$, and COL$: indexed by I_OBJ#
C_TS#	cluster containing FET$, TS$, and FILE$; indexed by I_TS#
C_USER#	cluster containing USER and TSQ$$; indexed by I_USER#
C_COBJ#	cluster containing CDEF$ and CCOL$; indexed by I_COBJ#

Of all of the data dictionary segments, the following are the most likely to require change:

C_TS#	if the free space in your database is very fragmented
C_OBJ#	if you have many indexes or many columns in your tables

CON$, C_COBJ#	if you use integrity constraints heavily
C_USER#	If you have a large number of users defined in your database

For the clustered tables, you must change the storage settings for the cluster, not for the table.

Errors that Require Changing Data Dictionary Storage

Oracle returns an error if a user tries to create a new object that requires Oracle to allocate an additional extent to the data dictionary when it is unable to allocate an extent. The error message ORA-1653, "failed to allocate extent of size *num* in tablespace '*name*'" indicates this kind of problem.

If you receive this error message and the segment you were trying to change (such as a table or rollback segment) has not reached the limits specified for it in its definition, check the storage settings for the object that contains its definition.

For example, if you received an ORA-1547 while trying to define a new PRIMARY KEY constraint on a table and there is sufficient space for the index that Oracle must create for the key, check if CON$ or C_COBJ# cannot be allocated another extent; to do this, query DBA_SEGMENTS and consider changing the storage parameters for CON$ or C_COBJ#.

See Also: For more information, see "Example 7: Displaying Segments that Cannot Allocate Additional Extents" on page 20-33.

Displaying Information About Schema Objects

The data dictionary provides many views about the schema objects described in this book. The following list summarizes the views associated with schema objects:

- ALL_OBJECTS, USER_OBJECTS, DBA_OBJECTS
- ALL_CATALOG, USER_CATALOG, DBA_CATALOG
- ALL_TABLES, USER_TABLES, DBA_TABLES
- ALL_TAB_COLUMNS, USER_TAB_COLUMNS, DBA_TAB_COLUMNS
- ALL_TAB_COMMENTS, USER_TAB_COMMENTS
- ALL_COL_COMMENTS, USER_COL_COMMENTS, DBA_COL_COMMENTS
- ALL_VIEWS, USER_VIEWS, DBA_VIEWS

- ALL_INDEXES, USER_INDEXES, DBA_INDEXES
- ALL_IND_COLUMNS, USER_IND_COLUMNS, DBA_IND_COLUMNS
- USER_CLUSTERS, DBA_CLUSTERS
- USER_CLU_COLUMNS, DBA_CLU_COLUMNS
- ALL_SEQUENCES, USER_SEQUENCES, DBA_SEQUENCES
- ALL_SYNONYMS, USER_SYNONYMS, DBA_SYNONYMS
- ALL_DEPENDENCIES, USER_DEPENDENCIES, DBA_DEPENDENCIES

The following data dictionary views contain information about the segments of a database:

- USER_SEGMENTS
- DBA_SEGMENTS

The following data dictionary views contain information about a database's extents:

- USER_EXTENTS
- DBA_EXTENTS
- USER_FREE_SPACE
- DBA_FREE_SPACE

Oracle Dictionary Storage Packages

Table 20–2 describes packages that are supplied with Oracle to either allow PL/SQL access to some SQL features, or to extend the functionality of the database.

Table 20–2 Supplied Packages: Additional Functionality

Procedure	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 blocks in an object (table, index, or cluster).
dbms_session.free_unused_user_memory	Procedure for reclaiming unused memory after performing operations requiring large amounts of memory (where large>100K). This procedure should only be used in cases where memory is at a premium.

The following examples demonstrate ways to display miscellaneous schema objects.

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;

OBJECT_NAME               OBJECT_TYPE
-------------------------  -------------------
EMP_DEPT                   CLUSTER
EMP                        TABLE
DEPT                       TABLE
EMP_DEPT_INDEX             INDEX
PUBLIC_EMP                 SYNONYM
EMP_MGR                    VIEW
```

Example 2: 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_default
    FROM user_tab_columns
    WHERE table_name = 'DEPT' OR table_name = 'EMP';

TABLE_NAME  COLUMN_NAME    DATA_DEFAULT
----------   ------------- --------------------
DEPT         DEPTNO
DEPT         DNAME
DEPT         LOC           'NEW YORK'
EMP          EMPNO
EMP          ENAME
EMP          JOB
EMP          MGR
EMP          HIREDATE      SYSDATE
EMP          SAL
EMP          COMM
EMP          DEPTNO
```

Notice that not all columns have user-specified defaults. These columns automatically have NULL as the default.

Example 3: 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/USER/DBA_DEPENDENCIES data dictionary views can be used to reveal the dependencies for a view and the ALL/USER/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 the user JWARD:

```
SELECT table_owner, table_name, synonym_name
    FROM sys.dba_synonyms
    WHERE owner = 'JWARD';


TABLE_OWNER             TABLE_NAME   SYNONYM_NAME
----------------------  -----------  -----------------
SCOTT                   DEPT         DEPT
SCOTT                   EMP          EMP
```

Example 4: Displaying General Segment Information

The following query returns the name of each rollback segment, the tablespace that contains each, and the size of each rollback segment:

```
SELECT segment_name, tablespace_name, bytes, blocks, extents
    FROM sys.dba_segments
    WHERE segment_type = 'ROLLBACK';

SEGMENT_NAME  TABLESPACE_NAME     BYTES       BLOCKS     EXTENTS
------------ ---------------      ---------   -------    ---------
RS1           SYSTEM              20480        10         2
RS2           TS1                 40960        20         3
SYSTEM        SYSTEM             184320        90         3
```

Example 5: Displaying General Extent Information

General 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 associated with rollback segments and the size of each of those extents:

```
SELECT segment_name, bytes, blocks
   FROM sys.dba_extents
    WHERE segment_type = 'ROLLBACK';
```

```
SEGMENT_NAME      BYTES         BLOCKS
---------------   ---------     --------
RS1               10240          5
RS1               10240          5
SYSTEM            51200         25
SYSTEM            51200         25
SYSTEM            51200         25
```

Notice that the RS1 rollback segment is comprised of two extents, both 10K, while the SYSTEM rollback segment is comprised of three equally sized extents of 50K.

Example 6: Displaying the Free Space (Extents) of a Database

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 via free extents in each tablespace:

```
SELECT tablespace_name, file_id, bytes, blocks
    FROM sys.dba_free_space;

TABLESPACE_NAME        FILE_ID     BYTES       BLOCKS
-------------------    ---------   --------    ----------
SYSTEM                 1           8120320      3965
SYSTEM                 1             10240         5
TS1                    2           10432512     5094
```

Example 7: Displaying Segments that Cannot Allocate Additional Extents

You can also use DBA_FREE_SPACE, in combination with the views DBA_SEGMENTS, DBA_TABLES, DBA_CLUSTERS, DBA_INDEXES, and DBA_ROLLBACK_SEGS, to determine if any other segment is unable to allocate additional extents for data dictionary objects only.

A segment may not 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, as recorded in the data dictionary (in SEG.MAX_EXTENTS).

- The segment has the maximum number of extents allowed by the data block size, which is operating system specific.

  Note: While the STORAGE clause value for MAXEXTENTS can be UNLIMITED, data dictionary tables cannot have MAXEXTENTS greater than the allowed block maximum. Thus, data dictionary tables cannot be converted to unlimited format.

The following query returns the names, owners, and tablespaces of all segments that fit any of the above criteria:

```
SELECT seg.owner, seg.segment_name,
    seg.segment_type, seg.tablespace_name,
    DECODE(seg.segment_type,
        'TABLE', t.next_extent,
        'CLUSTER', c.next_extent,
        'INDEX', i.next_extent,
        'ROLLBACK', r.next_extent)
FROM sys.dba_segments seg,
   sys.dba_tables t,
   sys.dba_clusters c,
   sys.dba_indexes i,
   sys.dba_rollback_segs r

WHERE ((seg.segment_type = 'TABLE'
   AND seg.segment_name = t.table_name
   AND seg.owner = t.owner
   AND NOT EXISTS (SELECT tablespace_name
       FROM dba_free_space free
       WHERE free.tablespace_name = t.tablespace_name
       AND free.bytes >= t.next_extent))
OR (seg.segment_type = 'CLUSTER'
    AND seg.segment_name = c.cluster_name
    AND seg.owner = c.owner
    AND NOT EXISTS (SELECT tablespace_name
       FROM dba_free_space free
       WHERE free.tablespace_name = c.tablespace_name
       AND free.bytes >= c.next_extent))
OR (seg.segment_type = 'INDEX'
    AND seg.segment_name = i.index_name
    AND seg.owner = i.owner
    AND NOT EXISTS (SELECT tablespace_name
    FROM dba_free_space free
       WHERE free.tablespace_name = i.tablespace_name
       AND free.bytes >= i.next_extent))
OR     (seg.segment_type = 'ROLLBACK'
    AND seg.segment_name = r.segment_name
```

```
    AND seg.owner = r.owner
    AND NOT EXISTS (SELECT tablespace_name
      FROM dba_free_space free
        WHERE free.tablespace_name = r.tablespace_name
    AND free.bytes >= r.next_extent)))
OR seg.extents = seg.max_extents OR seg.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 datafiles to the tablespace.
- If the segment has too many extents, and you cannot increase MAXEXTENTS for the segment, perform the following steps: first, export the data in the segment; second, drop and re-create the segment, giving it a larger INITIAL setting so that it does not need to allocate so many extents; and third, import the data back into the segment.

21

Managing Rollback Segments

This chapter describes how to manage rollback segments, and includes the following topics:

- Guidelines for Managing Rollback Segments
- Creating Rollback Segments
- Specifying Storage Parameters for Rollback Segments
- Taking Rollback Segments Online and Offline
- Explicitly Assigning a Transaction to a Rollback Segment
- Dropping Rollback Segments
- Monitoring Rollback Segment Information

See Also: If you are using Oracle with the Parallel Server option, see *Oracle8i Parallel Server Concepts and Administration*.

Guidelines for Managing Rollback Segments

This section describes guidelines to consider before creating or managing the rollback segments of your databases, and includes the following topics:

- Use Multiple Rollback Segments
- Choose Between Public and Private Rollback Segments
- Specify Rollback Segments to Acquire Automatically
- Set Rollback Segment Sizes Appropriately
- Create Rollback Segments with Many Equally Sized Extents
- Set an Optimal Number of Extents for Each Rollback Segment
- Set the Storage Location for Rollback

Every database contains one or more *rollback segments*, which are portions of the database that record the actions of transactions in the event that a transaction is rolled back. You use rollback segments to provide read consistency, roll back transactions, and recover the database.

See Also: For more information about rollback segments, see *Oracle8i Concepts*.

Use Multiple Rollback Segments

Using multiple rollback segments distributes rollback segment contention across many segments and improves system performance. Multiple rollback segments are required in the following situations:

- When a database is created, a single rollback segment named SYSTEM is created in the SYSTEM tablespace. You can create any objects in non-SYSTEM tablespaces, but you cannot write to them until you have created and brought online at least one additional rollback segment in a non- SYSTEM tablespace (for non-SYSTEM objects).
- When many transactions are concurrently proceeding, more rollback information is generated at the same time. You can indicate the number of concurrent transactions you expect for the instance with the parameter TRANSACTIONS, and the number of transactions you expect each rollback segment to have to handle with the parameter TRANSACTIONS_PER_ROLLBACK_SEGMENT. Then, when an instance opens a database, it attempts to acquire at least TRANSACTIONS / TRANSACTIONS_PER_ROLLBACK_SEGMENT rollback segments to handle the maximum amount of transactions. Therefore, after setting the parameters,

create TRANSACTIONS/TRANSACTIONS_PER_ROLLBACK_ SEGMENT rollback segments.

See Also: In order to start instances in an Oracle Parallel Server environment, you must give each instance access to its own rollback segment, in addition to the SYSTEM rollback segment. For additional details, see *Oracle8i Parallel Server Concepts and Administration*.

Add a Rollback Segment to the SYSTEM Tablespace

An instance always acquires the SYSTEM rollback segment in addition to any other rollback segments it needs. However, if there are multiple rollback segments, Oracle tries to use the SYSTEM rollback segment only for special system transactions and distributes user transactions among other rollback segments. If there are too many transactions for the non-SYSTEM rollback segments, Oracle uses the SYSTEM segment.

Choose Between Public and Private Rollback Segments

A *private rollback segment* is acquired explicitly by an instance when the instance opens the database. *Public rollback segments* form a pool of rollback segments that any instance requiring a rollback segment can use.

If a database does not have the Parallel Server option, public and private rollback segments are identical. Therefore, you can create all public rollback segments. A database with the Parallel Server option can also have only public segments, as long as the number of segments is high enough that each instance opening the database can acquire at least one rollback segment in addition to its SYSTEM rollback segment. You may also use private rollback segments when using the Oracle Parallel Server.

See Also: For more information about the Parallel Server option and rollback segments, see *Oracle8i Parallel Server Concepts and Administration*.

For more information about public and private rollback segments, see *Oracle8i Concepts*.

Specify Rollback Segments to Acquire Automatically

When an instance starts, it acquires by default TRANSACTIONS/ TRANSACTIONS_PER_ROLLBACK_SEGMENT rollback segments. If you want to ensure that the instance acquires particular rollback segments that have particular sizes or particular tablespaces, specify the rollback segments by name in the ROLLBACK_SEGMENTS parameter in the instance's parameter file.

The instance acquires all the rollback segments listed in this parameter, even if more than TRANSACTIONS / TRANSACTIONS_PER_ROLLBACK_SEGMENT segments are specified. The rollback segments can be either private or public.

Set Rollback Segment Sizes Appropriately

Total rollback segment size should be set based on the size of the most common transactions issued against a database. In general, short transactions experience better performance when the database has many smaller rollback segments, while long-running transactions, like batch jobs, perform better with larger rollback segments. Generally, rollback segments can handle transactions of any size easily; however, in extreme cases when a transaction is either very short or very long, a user might want to use an appropriately sized rollback segment.

If a system is running only short transactions, rollback segments should be small so that they are always cached in main memory. If the rollback segments are small enough, they are more likely to be cached in the SGA according to the LRU algorithm, and database performance is improved because less disk I / O is necessary. The main disadvantage of small rollback segments is the increased likelihood of the error "snapshot too old" when running a long query involving records that are frequently updated by other transactions. This error occurs because the rollback entries needed for read consistency are overwritten as other update entries wrap around the rollback segment. Consider this issue when designing an application's transactions, and make them short atomic units of work so that you can avoid this problem.

In contrast, long-running transactions work better with larger rollback segments, because the rollback entries for a long-running transaction can fit in preallocated extents of a large rollback segment.

When database systems applications concurrently issue a mix of very short and very long transactions, performance can be optimized if transactions are explicitly assigned to a rollback segment based on the transaction / rollback segment size. You can minimize dynamic extent allocation and truncation for rollback segments. This is not required for most systems and is intended for extremely large or small transactions.

To optimize performance when issuing a mix of extremely small and large transactions, make a number of rollback segments of appropriate size for each type of transaction (such as small, medium, and large). Most rollback segments should correspond to the typical transactions, with a fewer number of rollback segments for the atypical transactions. Then set OPTIMAL for each such rollback segment so that the rollback segment returns to its intended size if it has to grow.

You should tell users about the different sets of rollback segments that correspond to the different types of transactions. Often, it is *not* beneficial to assign a transaction explicitly to a specific rollback segment; however, you can assign an atypical transaction to an appropriate rollback segment created for such transactions. For example, you can assign a transaction that contains a large batch job to a large rollback segment.

When a mix of transactions is not prevalent, each rollback segment should be 10% of the size of the database's largest table because most SQL statements affect 10% or less of a table; therefore, a rollback segment of this size should be sufficient to store the actions performed by most SQL statements.

Generally speaking, you should set a high MAXEXTENTS for rollback segments; this allows a rollback segment to allocate subsequent extents as it needs them.

Create Rollback Segments with Many Equally Sized Extents

Each rollback segment's total allocated space should be divided among many equally sized extents. In general, optimal rollback I/O performance is observed if each rollback segment for an instance has 10 to 20 equally sized extents.

After determining the desired total initial size of a rollback segment and the number of initial extents for the segment, use the following formula to calculate the size of each extent of the rollback segment:

$T / n = s$

where:

T = total initial rollback segment size, in bytes

n = number of extents initially allocate

s = calculated size, in bytes, of each extent initially allocated

After *s* is calculated, create the rollback segment and specify the storage parameters INITIAL and NEXT as *s*, and MINEXTENTS to *n*. PCTINCREASE cannot be specified for rollback segments and therefore defaults to 0. Also, if the size *s* of an extent is not an exact multiple of the data block size, it is rounded up to the next multiple.

Set an Optimal Number of Extents for Each Rollback Segment

You should carefully assess the kind of transactions the system runs when setting the OPTIMAL parameter for each rollback segment. For a system that executes

long-running transactions frequently, OPTIMAL should be large so that Oracle does not have to shrink and allocate extents frequently. Also, for a system that executes long queries on active data, OPTIMAL should be large to avoid "snapshot too old" errors. OPTIMAL should be smaller for a system that mainly executes short transactions and queries so that the rollback segments remain small enough to be cached in memory, thus improving system performance.

To obtain estimates and monitor the effectiveness of the OPTIMAL settings for rollback segments, use the MONITOR ROLLBACK statement. The following statistics are given for each rollback segment:

Size, High Water	the most space ever allocated for the rollback segment, in bytes
Size, Optimal	the OPTIMAL size of the rollback segment, in bytes
Occurrences, Wraps	the cumulative number of times a transaction continues writing from one extent in a rollback segment to another existing extent
Occurrences, Extends	the cumulative number of times a new extent is allocated for a rollback segment
Shrinks	the cumulative number of times Oracle has truncated extents from the rollback segment
Average Size, Shrunk	the average size of the space Oracle truncated from the rollback segment, in bytes
Average Size, Active	the average number of bytes in active extents in the rollback segment, measured over time

Assuming that an instance has equally sized rollback segments with comparably sized extents, the OPTIMAL parameter for a given rollback segment should be set

slightly higher than *Average Sizes, Active*. Table 21–1 provides additional information on how to interpret the statistics given in this monitor.

Table 21–1 Analyzing the Effectiveness of Current OPTIMAL Settings

Shrinks	Average Sizes, Shrunk	Analysis and Recommendation
Low	Low	If *Average Sizes, active* is close to *Sizes, Optimal*, then the OPTIMAL setting is correct. Otherwise, OPTIMAL is too large (not many shrinks are being performed.)
Low	High	Excellent: a good setting for OPTIMAL.
High	Low	OPTIMAL is too small: too many shrinks are being performed.
High	High	Periodic long transactions are probably causing these statistics. Set the OPTIMAL parameter higher until *Shrinks* is low.

Set the Storage Location for Rollback

If possible, create one tablespace specifically to hold all rollback segments. This way, all rollback segment data is stored separately from other types of data. Creating this "rollback segment" tablespace can provide the following benefits:

- A tablespace holding rollback segments can always be kept online, thus maximizing the combined storage capacity of rollback segments at all times. Note that if some rollback segments are not available, the overall database operation can be affected.
- Because tablespaces with active rollback segments cannot be taken offline, designating a tablespace to hold all rollback segments of a database ensures that the data stored in other tablespaces can be taken offline without concern for the database's rollback segments.
- A tablespace's free extents are likely to be more fragmented if the tablespace contains rollback segments that frequently allocate and deallocate extents.

Creating Rollback Segments

To create rollback segments, you must have the CREATE ROLLBACK SEGMENT system privilege. To create additional rollback segments for a database, use the SQL

statement CREATE ROLLBACK SEGMENT. The tablespace to contain the new rollback segment must be online.

The following statement creates a public rollback segment named USERS_RS in the USERS tablespace, using the default storage parameters of the USERS tablespace:

```
CREATE PUBLIC ROLLBACK SEGMENT users_rs TABLESPACE users;
```

Bringing New Rollback Segments Online

If you create a private rollback segment, you should add the name of this new rollback segment to the ROLLBACK_SEGMENTS parameter in the parameter file for the database. Doing so enables the private rollback segment to be captured by the instance at instance start up. For example, if two new private rollback segments are created and named RS1 and RS2, the ROLLBACK_SEGMENTS parameter of the parameter file should be similar to the following:

```
ROLLBACK SEGMENTS= (RS1, RS2)
```

See Also: Once a rollback segment is created, it is not available for use by transactions of any instance until it is brought online. See "Taking Rollback Segments Online and Offline" on page 21-10 for more information.

Specifying Storage Parameters for Rollback Segments

This section describes aspects of specifying rollback segment storage parameters, and includes the following topics:

- Setting Storage Parameters When Creating a Rollback Segment
- Changing Rollback Segment Storage Parameters
- Altering Rollback Segment Format
- Shrinking a Rollback Segment Manually

Setting Storage Parameters When Creating a Rollback Segment

Suppose you wanted to create a public rollback segment DATA1_RS with storage parameters and optimal size set as follows:

- The rollback segment is allocated an initial extent of 50K.
- The rollback segment is allocated the second extent of 50K.
- The optimal size of the rollback segment is 750K.

- The minimum number of extents and the number of extents initially allocated when the segment is created is 15.
- The maximum number of extents that the rollback segment can allocate, including the initial extent, is 100.

The following statement creates a rollback segment with these characteristics:

```
CREATE PUBLIC ROLLBACK SEGMENT data1_rs
                TABLESPACE users
                STORAGE (
      INITIAL 50K
      NEXT 50K
      OPTIMAL 750K
      MINEXTENTS 15
      MAXEXTENTS 100);
```

Changing Rollback Segment Storage Parameters

You can change a rollback segment's storage parameters after creating it. However, you cannot alter the size of any extent currently allocated to a rollback segment. You can only affect future extents.

Alter a rollback segment's storage parameters using the SQL statement ALTER ROLLBACK SEGMENT.

The following statement alters the maximum number of extents that the DATA1_RS rollback segment can allocate.

```
ALTER PUBLIC ROLLBACK SEGMENT data1_rs
STORAGE (MAXEXTENTS 120);
```

You can alter the settings for the SYSTEM rollback segment, including the OPTIMAL parameter, just as you can alter those of any rollback segment.

See Also: For guidance on setting sizes and storage parameters (including OPTIMAL) for rollback segments, see "Guidelines for Managing Rollback Segments" on page 21-2.

Altering Rollback Segment Format

To alter rollback segments, you must have the ALTER ROLLBACK SEGMENT system privilege.

You can define limited or unlimited format for rollback segments. When converting to limited or unlimited format, you *must* take the rollback segments offline. If you

identify unlimited format for rollback segments, extents for that segment must have a minimum of 4 data blocks. Thus, a limited format rollback segment cannot be converted to unlimited format if it has less than 4 data blocks in any extent. If you want to convert from limited to unlimited format and have less than 4 data blocks in an extent, your only choice is to drop and re-create the rollback segment.

Shrinking a Rollback Segment Manually

To shrink a rollback segment you must have the ALTER ROLLBACK SEGMENT system privilege.

You can manually decrease the size of a rollback segment using the SQL statement ALTER ROLLBACK SEGMENT. The rollback segment you are trying shrink must be online.

The following statement shrinks rollback segment RBS1 to 100K:

```
ALTER ROLLBACK SEGMENT rbs1 SHRINK TO 100K;
```

See Also: For a complete description of the ALTER ROLLBACK SEGMENT statement, see the *Oracle8i SQL Reference*.

Taking Rollback Segments Online and Offline

This section describes aspects of taking rollback segments online and offline, and includes the following topics:

- Bringing Rollback Segments Online
- Taking Rollback Segments Offline

A rollback segment is either *online* and available to transactions, or *offline* and unavailable to transactions. Generally, rollback segments are online and available for use by transactions.

You may wish to take online rollback segments offline in the following situations:

- When you want to take a tablespace offline, and the tablespace contains rollback segments. You cannot take a tablespace offline if it contains rollback segments that transactions are currently using. To prevent associated rollback segments from being used, you can take them offline before taking the tablespace offline.

- You want to drop a rollback segment, but cannot because transactions are currently using it. To prevent the rollback segment from being used, you can take it offline before dropping it.

  Note: You cannot take the SYSTEM rollback segment offline.

You might later want to bring an offline rollback segment back online so that transactions can use it. When a rollback segment is created, it is initially offline, and you must explicitly bring a newly created rollback segment online before it can be used by an instance's transactions. You can bring an offline rollback segment online via any instance accessing the database that contains the rollback segment.

Bringing Rollback Segments Online

You can bring online only a rollback segment whose current status (as shown in the DBA_ROLLBACK_SEGS data dictionary view) is OFFLINE or PARTLY AVAILABLE. To bring an offline rollback segment online, use the SQL statement ALTER ROLLBACK SEGMENT with the ONLINE option.

Bringing a PARTLY AVAILABLE Rollback Segment Online

A rollback segment in the PARTLY AVAILABLE state contains data for an in-doubt or recovered distributed transaction, and yet to be recovered transactions. You can view its status in the data dictionary view DBA_ROLLBACK_SEGS as PARTLY AVAILABLE. The rollback segment usually remains in this state until the transaction is resolved either automatically by RECO, or manually by a DBA. However, you might find that all rollback segments are PARTLY AVAILABLE. In this case, you can bring a PARTLY AVAILABLE segment online, as described above.

Some resources used by the rollback segment for the in-doubt transaction remain inaccessible until the transaction is resolved. As a result, the rollback segment may have to grow if other transactions assigned to it need additional space.

As an alternative to bringing a PARTLY AVAILABLE segment online, you might find it easier to create a new rollback segment temporarily, until the in-doubt transaction is resolved.

Bringing Rollback Segment Online Automatically

If you would like a rollback segment to be automatically brought online whenever you start up the database, add the segment's name to the ROLLBACK_SEGMENTS parameter in the database's parameter file.

Bringing Rollback Segments Online: Example

The following statement brings the rollback segment USER_RS_2 online:

```
ALTER ROLLBACK SEGMENT user_rs_2 ONLINE;
```

After you bring a rollback segment online, its status in the data dictionary view DBA_ROLLBACK_SEGS is ONLINE.

See Also: For information about the ROLLBACK_SEGMENTS and DBA_ROLLBACK_SEGS parameters, see the *Oracle8i Reference*.

To see a query for checking rollback segment state, see "Displaying Rollback Segment Information" on page 21-14.

Taking Rollback Segments Offline

To take an online rollback segment offline, use the ALTER ROLLBACK SEGMENT command with the OFFLINE option. The rollback segment's status in the DBA_ROLLBACK_SEGS data dictionary view must be "ONLINE", and the rollback segment must be acquired by the current instance.

The following example takes the rollback segment USER_RS_2 offline:

```
ALTER ROLLBACK SEGMENT user_rs_2 OFFLINE;
```

If you try to take a rollback segment that does not contain active rollback entries offline, Oracle immediately takes the segment offline and changes its status to "OFFLINE".

In contrast, if you try to take a rollback segment that contains rollback data for active transactions (local, remote, or distributed) offline, Oracle makes the rollback segment unavailable to future transactions and takes it offline after all the active transactions using the rollback segment complete. Until the transactions complete, the rollback segment cannot be brought online by any instance other than the one that was trying to take it offline. During this period, the rollback segment's status in the view DBA_ROLLBACK_SEGS remains ONLINE; however, the rollback segment's status in the view V$ROLLSTAT is PENDING OFFLINE.

The instance that tried to take a rollback segment offline and caused it to change to PENDING OFFLINE can bring it back online at any time; if the rollback segment is brought back online, it will function normally.

Taking Public and Private Rollback Segments Offline

After you take a public or private rollback segment offline, it remains offline until you explicitly bring it back online *or* you restart the instance.

See Also: For information on viewing rollback segment status, see "Displaying Rollback Segment Information" on page 21-14.

For information about the views DBA_ROLLBACK_SEGS and V$ROLLSTAT, see the *Oracle8i Reference*.

Explicitly Assigning a Transaction to a Rollback Segment

A transaction can be explicitly assigned to a specific rollback segment using the SET TRANSACTION statement with the USE ROLLBACK SEGMENT clause. Transactions are explicitly assigned to rollback segments for the following reasons:

- The anticipated amount of rollback information generated by a transaction can fit in the current extents of the assigned rollback segment.
- Additional extents do not have to be dynamically allocated (and subsequently truncated) for rollback segments, which reduces overall system performance.

To assign a transaction to a rollback segment explicitly, the rollback segment must be online for the current instance, and the SET TRANSACTION USE ROLLBACK SEGMENT statement must be the first statement of the transaction. If a specified rollback segment is not online or a SET TRANSACTION USE ROLLBACK SEGMENT clause is not the first statement in a transaction, an error is returned.

For example, if you are about to begin a transaction that contains a significant amount of work (more than most transactions), you can assign the transaction to a large rollback segment, as follows:

```
SET TRANSACTION USE ROLLBACK SEGMENT large_rs1;
```

After the transaction is committed, Oracle will automatically assign the next transaction to any available rollback segment unless the new transaction is explicitly assigned to a specific rollback segment by the user.

Dropping Rollback Segments

You can drop rollback segments when the extents of a segment become too fragmented on disk, or the segment needs to be relocated in a different tablespace.

Before dropping a rollback segment, make sure that status of the rollback segment is OFFLINE. If the rollback segment that you want to drop is currently ONLINE, PARTLY AVAILABLE, NEEDS RECOVERY, or INVALID, you cannot drop it. If the status is INVALID, the segment has already been dropped. Before you can drop it, you must take it offline.

To drop a rollback segment, you must have the DROP ROLLBACK SEGMENT system privilege.

If a rollback segment is offline, you can drop it using the SQL statement DROP ROLLBACK SEGMENT.

The following statement drops the DATA1_RS rollback segment:

```
DROP PUBLIC ROLLBACK SEGMENT data1_rs;
```

If you use the DROP ROLLBACK SEGMENT statement, indicate the correct type of rollback segment to drop, public or private, by including or omitting the PUBLIC keyword.

> **Note:** If a rollback segment specified in ROLLBACK_SEGMENTS is dropped, make sure to edit the parameter files of the database to remove the name of the dropped rollback segment from the list in the ROLLBACK_SEGMENTS parameter. If this step is not performed before the next instance startup, startup fails because it cannot acquire the dropped rollback segment.

After a rollback segment is dropped, its status changes to INVALID. The next time a rollback segment is created, it takes the row vacated by a dropped rollback segment, if one is available, and the dropped rollback segment's row no longer appears in the DBA_ROLLBACK_SEGS view.

See Also: For more information about the view DBA_ROLLBACK_SEGS, see the *Oracle8i Reference*.

Monitoring Rollback Segment Information

For a detailed description of how to use the MONITOR for the corresponding operation, see "Set an Optimal Number of Extents for Each Rollback Segment" on page 21-5.

Displaying Rollback Segment Information

The DBA_ROLLBACK_SEGS data dictionary view stores information about the rollback segments of a database. For example, the following query lists the name, associated tablespace, and status of each rollback segment in a database:

```
SELECT segment_name, tablespace_name, status
       FROM sys.dba_rollback_segs;
```

```
SEGMENT_NAME  TABLESPACE_NAME     STATUS
------------- ----------------    ------
SYSTEM        SYSTEM              ONLINE
PUBLIC_RS     SYSTEM              ONLINE
USERS_RS      USERS               ONLINE
```

In addition, the following data dictionary views contain information about the segments of a database, including rollback segments:

- USER_SEGMENTS
- DBA_SEGMENTS

Displaying All Rollback Segments

The following query returns the name of each rollback segment, the tablespace that contains it, and its size:

```
SELECT segment_name, tablespace_name, bytes, blocks, extents
        FROM sys.dba_segments
   WHERE segment_type = 'ROLLBACK';

SEGMENT_NAME   TABLESPACE_NAME    BYTES      BLOCKS     EXTENTS
------------   ---------------    -------    ------     ---------
RS1            SYSTEM              20480        10         2
RS2            TS1                 40960        20         3
SYSTEM         SYSTEM             184320        90         3
```

Displaying Whether a Rollback Segment Has Gone Offline

When you take a rollback segment offline, it does not actually go offline until all active transactions in it have completed. Between the time when you attempt to take it offline and when it actually is offline, its status in DBA_ROLLBACK_SEGS remains ONLINE, but it is not used for new transactions. To determine whether any rollback segments for an instance are in this state, use the following query:

```
SELECT name, xacts 'ACTIVE TRANSACTIONS'
      FROM v$rollname, v$rollstat
WHERE status = 'PENDING OFFLINE'
      AND v$rollname.usn = v$rollstat.usn;

NAME         ACTIVE TRANSACTIONS
----------   --------------------
RS2             3
```

If your instance is part of a Parallel Server configuration, this query displays information for rollback segments of the current instance only, not those of other instances.

Displaying Deferred Rollback Segments

The following query shows which rollback segments are private and which are public. Note that it only displays information about the rollback segments that are currently online for the current instance:

```
SELECT segment_name, tablespace_name, owner
       FROM sys.dba_rollback_segs;

SEGMENT_NAME   TABLESPACE_NAME    OWNER
-------------  ----------------   ------
SYSTEM         SYSTEM             SYS
PUBLIC_RS      SYSTEM             PUBLIC
USERS_RS       USERS              SYS
```

Displaying All Deferred Rollback Segments

The following query shows all deferred rollback segments (rollback segments that were created to hold rollback entries for tablespaces taken offline until the tablespaces are brought back online):

```
SELECT segment_name, segment_type, tablespace_name
     FROM sys.dba_segments
WHERE segment_type = 'DEFERRED ROLLBACK';

SEGMENT_NAME     SEGMENT_TYPE        TABLESPACE_NAME
------------     -----------------   ----------------
USERS_RS         DEFERRED ROLLBACK   USERS
```

Part IV

Database Security

22

Establishing Security Policies

This chapter provides guidelines for developing security policies for database operation, and includes the following topics:

- System Security Policy
- Data Security Policy
- User Security Policy
- Password Management Policy
- Auditing Policy

System Security Policy

This section describes aspects of system security policy, and includes the following topics:

- Database User Management
- User Authentication
- Operating System Security

Each database has one or more administrators who are responsible for maintaining all aspects of the security policy: the security administrators. If the database system is small, the database administrator may have the responsibilities of the security administrator. However, if the database system is large, a special person or group of people may have responsibilities limited to those of a security administrator.

After deciding who will manage the security of the system, a security policy must be developed for every database. A database's security policy should include several sub-policies, as explained in the following sections.

Database User Management

Database users are the access paths to the information in an Oracle database. Therefore, tight security should be maintained for the management of database users. Depending on the size of a database system and the amount of work required to manage database users, the security administrator may be the only user with the privileges required to create, alter, or drop database users. On the other hand, there may be a number of administrators with privileges to manage database users. Regardless, only trusted individuals should have the powerful privileges to administer database users.

User Authentication

Database users can be *authenticated* (verified as the correct person) by Oracle using the host operating system, network services, or the database. Generally, user authentication via the host operating system is preferred for the following reasons:

- Users can connect to Oracle faster and more conveniently without specifying a username or password.
- Centralized control over user authorization in the operating system: Oracle need not store or manage user passwords and usernames if the operating system and database correspond.
- User entries in the database and operating system audit trails correspond.

User authentication by the database is normally used when the host operating system cannot support user authentication.

See Also: For more information about network authentication, see *Oracle8i Distributed Database Systems.*

For more information about user authentication, see "Creating Users" on page 23-11.

Operating System Security

If applicable, the following security issues must also be considered for the operating system environment executing Oracle and any database applications:

- Database administrators must have the operating system privileges to create and delete files.
- Typical database users should not have the operating system privileges to create or delete files related to the database.
- If the operating system identifies database roles for users, the security administrators must have the operating system privileges to modify the security domain of operating system accounts.

See Also: For more information about operating system security issues for Oracle databases, see your operating system-specific Oracle documentation.

Data Security Policy

Data security includes the mechanisms that control the access to and use of the database at the object level. Fine-grained access control can also limit data access to a more granular level. Your data security policy determines which users have access to a specific schema object, and the specific types of actions allowed for each user on the object. For example, user SCOTT can issue SELECT and INSERT statements but not DELETE statements using the EMP table. Your data security policy should also define the actions, if any, that are audited for each schema object.

Your data security policy will be determined primarily by the level of security you wish to establish for the data in your database. For example, it may be acceptable to have little data security in a database when you wish to allow any user to create any schema object, or grant access privileges for their objects to any other user of the system. Alternatively, it might be necessary for data security to be very controlled when you wish to make a database or security administrator the only person with the privileges to create objects and grant access privileges for objects to roles and users.

Overall data security should be based on the sensitivity of data. If information is not sensitive, then the data security policy can be more lax. However, if data is sensitive, a security policy should be developed to maintain tight control over access to objects.

User Security Policy

This section describes aspects of user security policy, and includes the following topics:

- General User Security
- End-User Security
- Administrator Security
- Application Developer Security
- Application Administrator Security

General User Security

For all types of database users, consider the following general user security issues:

- Password Security
- Privilege Management

Password Security

If user authentication is managed by the database, security administrators should develop a password security policy to maintain database access security. For example, database users should be required to change their passwords at regular intervals, and of course, when their passwords are revealed to others. By forcing a user to modify passwords in such situations, unauthorized database access can be reduced.

Secure Connections with Encrypted Passwords

To better protect the confidentiality of your password, Oracle can be configured to use encrypted passwords for client/server and server/server connections.

By setting the following values, you can require that the password used to verify a connection always be encrypted:

- Set the ORA_ENCRYPT_LOGIN environment variable to TRUE on the client machine.
- Set the DBLINK_ENCRYPT_LOGIN server initialization parameter to TRUE.

If enabled at both the client and server, passwords will not be sent across the network "in the clear", but will be encrypted using a modified DES (Data Encryption Standard) algorithm.

The DBLINK_ENCRYPT_LOGIN parameter is used for connections between two Oracle servers (for example, when performing distributed queries). If you are connecting from a client, Oracle checks the ORA_ENCRYPT_LOGIN environment variable.

Whenever you attempt to connect to a server using a password, Oracle encrypts the password before sending it to the server. If the connection fails and auditing is enabled, the failure is noted in the audit log. Oracle then checks the appropriate DBLINK_ENCRYPT_LOGIN or ORA_ENCRYPT_LOGIN value. If it set to FALSE, Oracle attempts the connection again using an unencrypted version of the password. If the connection is successful, the connection replaces the previous failure in the audit log, and the connection proceeds. To prevent malicious users from forcing Oracle to re-attempt a connection with an unencrypted version of the password, you must set the appropriate values to TRUE.

Privilege Management

Security administrators should consider issues related to privilege management for all types of users. For example, in a database with many usernames, it may be beneficial to use roles (which are named groups of related privileges that you grant to users or other roles) to manage the privileges available to users. Alternatively, in a database with a handful of usernames, it may be easier to grant privileges explicitly to users and avoid the use of roles.

Security administrators managing a database with many users, applications, or objects should take advantage of the benefits offered by roles. Roles greatly simplify the task of privilege management in complicated environments.

End-User Security

Security administrators must also define a policy for end-user security. If a database is large with many users, the security administrator can decide what groups of users can be categorized, create user roles for these user groups, grant the necessary privileges or application roles to each user role, and assign the user roles to the

users. To account for exceptions, the security administrator must also decide what privileges must be explicitly granted to individual users.

Using Roles for End-User Privilege Management

Roles are the easiest way to grant and manage the common privileges needed by different groups of database users.

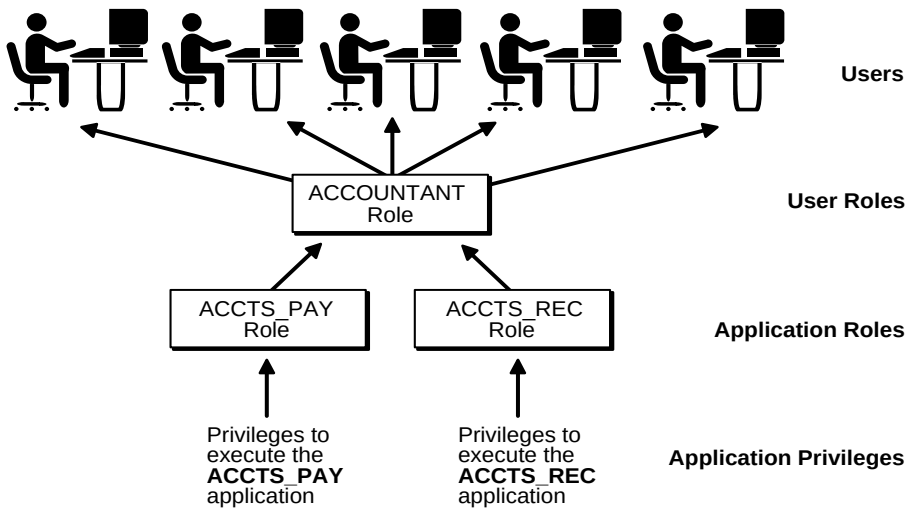
Consider a situation where every user in the accounting department of a company needs the privileges to run the ACCTS_RECEIVABLE and ACCTS_PAYABLE database applications. Roles are associated with both applications, and contain the object privileges necessary to execute those applications.

The following actions, performed by the database or security administrator, address this simple security situation:

1. Create a role named ACCOUNTANT.
2. Grant the roles for the ACCTS_RECEIVABLE and ACCTS_PAYABLE database applications to the ACCOUNTANT role.
3. Grant each user of the accounting department the ACCOUNTANT role.

This security model is illustrated in Figure 22–1.

Figure 22–1 User Role



This plan addresses the following potential situations:

- If accountants subsequently need a role for a new database application, that application's role can be granted to the ACCOUNTANT role, and all users in the accounting department will automatically receive the privileges associated with the new database application. The application's role does not need to be granted to individual users requiring use of the application.
- Similarly, if the accounting department no longer requires the need for a specific application, the application's role can be dropped from the ACCOUNTANT role.
- If the privileges required by the ACCTS_RECEIVABLE or ACCTS_PAYABLE applications change, the new privileges can be granted to, or revoked from, the application's role. The security domain of the ACCOUNTANT role, and all users granted the ACCOUNTANT role automatically reflect the privilege modification.
- You have an index where a user requires only 1 role.

When possible, utilize roles in all possible situations to make end-user privilege management efficient and simple.

Administrator Security

Security administrators should have a policy addressing administrator security. For example, when the database is large and there are several types of database administrators, the security administrator may decide to group related administrative privileges into several administrative roles. The administrative roles can then be granted to appropriate administrator users. Alternatively, when the database is small and has only a few administrators, it may be more convenient to create one administrative role and grant it to all administrators.

Protection for Connections as SYS and SYSTEM

After database creation, *immediately* change the passwords for the administrative SYS and SYSTEM usernames to prevent unauthorized access to the database. Connecting as SYS and SYSTEM gives a user the powerful privileges to modify a database in many ways. Therefore, privileges for these usernames are extremely sensitive, and should only be available to select database administrators.

See Also: The passwords for these accounts can be modified using the procedures described in "Altering Users" on page 23-15.

Protection for Administrator Connections

Only database administrators should have the capability to connect to a database with administrator privileges. Connecting as SYSDBA gives a user unrestricted privileges to do anything to a database (such as startup, shutdown, and recover) or the objects within a database (such as create, drop, and delete from). Only users with SYS-privileged connections can alter data dictionary tables (for example, `connect as SYSDBA/SYSOPER`).

Using Roles for Administrator Privilege Management

Roles are the easiest way to restrict the powerful system privileges and roles required by personnel administrating of the database.

Consider a scenario where the database administrator responsibilities at a large installation are shared among several database administrators, each responsible for the following specific database management jobs:

- an administrator responsible for object creation and maintenance
- an administrator responsible for database tuning and performance
- a security administrator responsible for creating new users, granting roles and privileges to database users
- a database administrator responsible for routine database operation (for example, startup, shutdown, backup)
- an administrator responsible for emergency situations, such as database recovery
- new, inexperienced database administrators needing limited capabilities to experiment with database management

In this scenario, the security administrator should structure the security for administrative personnel as follows:

1. Six roles should be defined to contain the distinct privileges required to accomplish each type of job (for example, DBA_OBJECTS, DBA_TUNE, DBA_SECURITY, DBA_MAINTAIN, DBA_RECOV, DBA_NEW).
2. Each role is granted the appropriate privileges.
3. Each type of database administrator can be granted the corresponding role.

This plan diminishes the likelihood of future problems in the following ways:

- If a database administrator's job description changes to include more responsibilities, that database administrator can be granted other administrative roles corresponding to the new responsibilities.
- If a database administrator's job description changes to include fewer responsibilities, that database administrator can have the appropriate administrative roles revoked.
- The data dictionary always stores information about each role and each user, so information is available to disclose the task of each administrator.

Application Developer Security

Security administrators must define a special security policy for the application developers using a database. A security administrator may grant the privileges to create necessary objects to application developers. Alternatively, the privileges to create objects may only be granted to a database administrator, who receives requests for object creation from developers.

Application Developers and Their Privileges

Database application developers are unique database users who require special groups of privileges to accomplish their jobs. Unlike end users, developers need system privileges, such as CREATE TABLE, CREATE PROCEDURE, and so on. However, only specific system privileges should be granted to developers to restrict their overall capabilities in the database.

The Application Developer's Environment: Test and Production Databases

In many cases, application development is restricted to test databases and not allowed on production databases. This restriction ensures that application developers do not compete with end users for database resources, and that they cannot detrimentally affect a production database.

After an application has been thoroughly developed and tested, it is permitted access to the production database and made available to the appropriate end users of the production database.

Free Versus Controlled Application Development

The database administrator can define the following options when determining which privileges should be granted to application developers:

Free Development	An application developer is allowed to create new schema objects, including tables, indexes, procedures, packages, and so on. This option allows the application developer to develop an application independent of other objects.
Controlled Development	An application developer is not allowed to create new schema objects. All required tables, indexes, procedures, and so on are created by a database administrator, as requested by an application developer. This option allows the database administrator to completely control a database's space usage and the access paths to information in the database.

Although some database systems use only one of these options, other systems could mix them. For example, application developers can be allowed to create new stored procedures and packages, but not allowed to create tables or indexes. A security administrator's decision regarding this issue should be based on the following:

- the control desired over a database's space usage
- the control desired over the access paths to schema objects
- the database used to develop applications—if a test database is being used for application development, a more liberal development policy would be in order

Roles and Privileges for Application Developers

Security administrators can create roles to manage the privileges required by the typical application developer. For example, a typical role named APPLICATION_DEVELOPER might include the CREATE TABLE, CREATE VIEW, and CREATE PROCEDURE system privileges. Consider the following when defining roles for application developers:

- CREATE system privileges are usually granted to application developers so that they can create their own objects. However, CREATE ANY system privileges, which allow a user to create an object in any user's schema, are not usually granted to developers. This restricts the creation of new objects only to the developer's user account.
- Object privileges are rarely granted to roles used by application developers. This is often impractical because granting object privileges via roles often restricts their usability in the creation of other objects (primarily views and

stored procedures). It is more practical to allow application developers to create their own objects for development purposes.

Space Restrictions Imposed on Application Developers

While application developers are typically given the privileges to create objects as part of the development process, security administrators must maintain limits on what and how much database space can be used by each application developer. For example, as the security administrator, you should specifically set or restrict the following limits for each application developer:

- the tablespaces in which the developer can create tables or indexes
- the quota for each tablespace accessible to the developer

See Also: Both limitations can be set by altering a developer's security domain. For more information, see "Altering Users" on page 23-15.

Application Administrator Security

In large database systems with many database applications (for example, precompiler and Forms applications), you might want to have application administrators. An application administrator is responsible for the following types of tasks:

- creating roles for an application and managing the privileges of each application role
- creating and managing the objects used by a database application
- maintaining and updating the application code and Oracle procedures and packages, as necessary

Often, an application administrator is also the application developer who designed the application. However, these jobs might not be the responsibility of the developer and can be assigned to another individual familiar with the database application.

Password Management Policy

Database security systems depend on passwords being kept secret at all times. Still, passwords are vulnerable to theft, forgery, and misuse.To allow for greater control over database security, Oracle's password management policy is controlled by DBAs.

This section describes the following aspects of Oracle password management:

- Account Locking
- Password Aging and Expiration
- Password History
- Password Complexity Verification

Account Locking

When a particular user exceeds a designated number of failed login attempts, the server automatically locks that user's account. DBAs specify the permissible number of failed login attempts using the CREATE PROFILE statement. DBAs also specify the amount of time accounts remain locked.

In the following example, the maximum number of failed login attempts for the user ASHWINI is 4, and the amount of time the account will remain locked is 30 days; the account will unlock automatically after the passage of 30 days.

```
CREATE PROFILE prof LIMIT
   FAILED_LOGIN_ATTEMPTS 4
    PASSWORD_LOCK_TIME 30;
ALTER USER ashwini PROFILE prof;
```

If the DBA does not specify a time interval for unlocking the account, ACCOUNT_LOCK _TIME reverts to a default value. If the DBA specifies ACCOUNT_LOCK_TIME as UNLIMITED, then the system security officer must explicitly unlock the account. Thus, the amount of time an account remains locked depends upon how the DBA configures the resource profile assigned to the user.

After a user successfully logs into an account, that user's unsuccessful login attempt count, if there is one, is reset to 0.

The security officer can also explicitly lock user accounts. When this occurs, the account cannot be unlocked automatically; only the security officer should unlock the account.

See Also: For more information about the CREATE PROFILE statement, see the *Oracle8i SQL Reference.*

Password Aging and Expiration

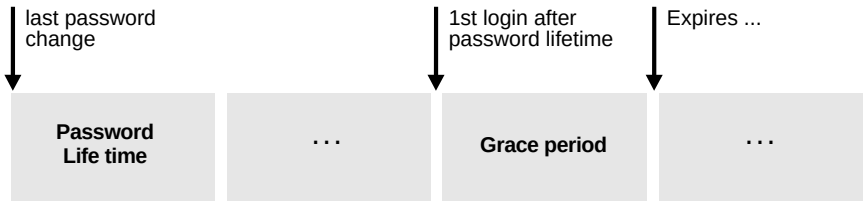
DBAs use the CREATE PROFILE statement to specify a maximum lifetime for passwords. When the specified amount of time passes and the password expires,

the user or DBA must change the password. The following statement indicates that ASHWINI can use the same password for 90 days before it expires:

```
CREATE PROFILE prof LIMIT
   FAILED_LOGIN_ATTEMPTS 4
   PASSWORD_LOCK_TIME 30
   PASSWORD_LIFE_TIME 90;
ALTER USER ashwini PROFILE prof;
```

DBAs can also specify a grace period using the CREATE PROFILE statement. Users enter the grace period upon the first attempt to login to a database account after their password has expired. During the grace period, a warning message appears each time users try to log in to their accounts, and continues to appear until the grace period expires. Users must change the password within the grace period. If the password is not changed within the grace period, the account expires and no further logins to that account are allowed until the password is changed. Figure 22–2 shows the chronology of the password lifetime and grace period.

Figure 22–2 Chronology of Password Lifetime and Grace Period.



For example, the lifetime of a password is 60 days, and the grace period is 3 days. If the user tries to log in on *any* day after the 60th day (this could be the 70th day, 100th day, or another; the point here is that it is the first login attempt after the password lifetime), that user receives a warning message indicating that the password is about to expire in 3 days. If the user does not change the password within three days from the first day of the grace period, the user's account expires. The following statement indicates that the user must change the password within 3 days of its expiration:

```
CREATE PROFILE prof LIMIT
   FAILED_LOGIN_ATTEMPTS 4
   ACCOUNT_LOCK_TIME 30
   PASSWORD_GRACE_TIME 3;
ALTER USER ashwini PROFILE prof;
```

The security officer can also explicitly expire the account. This is particularly useful for new accounts.

See Also: For more information about the CREATE PROFILE statement, see *Oracle8i SQL Reference.*

Password History

DBAs use the CREATE PROFILE statement to specify a time interval during which users cannot reuse a password.

In the following statement, the DBA indicates that the user cannot reuse her password for 60 days.

```
CREATE PROFILE prof LIMIT
   PASSWORD_REUSE_TIME 60
   PASSWORD_REUSE_MAX UNLIMITED;
```

The next statement shows that the number of password changes the user must make before her current password can be used again is 3.

```
CREATE PROFILE prof LIMIT
PASSWORD_REUSE_MAX 3
PASSWORD_REUSE_TIME UNLIMITED;
```

Note: If you specify PASSWORD_REUSE_TIME or PASSWORD_REUSE_MAX, you must set the other to UNLIMITED or not specify it at all.

Password Complexity Verification

Oracle's password complexity verification routine can be specified using a PL/SQL script (utlpwdmg.sql), which sets the default profile parameters.

The password complexity verification routine performs the following checks:

- The password has a minimum length of 4.
- The password is not the same as the userid.
- The password has at least one alpha, one numeric, and one punctuation mark.
- The password does not match simple words like welcome, account, database, or user.

- The password differs from the previous password by at least 3 letters.

> **Note:** Oracle recommends that you do not change passwords using the ALTER USER statement because it does not fully support the password verification function. Instead, you should use OCIPasswordChange() to change passwords.

Password Verification Routine Formatting Guidelines

DBAs can enhance the existing password verification complexity routine or create their own password verification routines using PL/SQL or third-party tools.

The DBA-authored PL/SQL call must adhere to the following format:

```
routine_name (
userid_parameter IN VARCHAR(30),
password_parameter IN VARCHAR (30),
old_password_parameter IN VARCHAR (30)
)
RETURN BOOLEAN
```

After a new routine is created, it must be assigned as the password verification routine using the user's profile or the system default profile.

```
CREATE/ALTER PROFILE profile_name LIMIT
PASSWORD_VERIFY_FUNCTION routine_name
```

The password verify routine must be owned by SYS.

Password Verification Routine: Sample Script The following sample script sets default password resource limits and provides minimum checking of password complexity. You can use this sample script as a model when developing your own complexity checks for a new password.

This script sets the default password resource parameters, and must be run to enable the password features. However, you can change the default resource parameters if necessary.

The default password complexity function performs the following minimum complexity checks:

- The password satisfies minimum length requirements.
- The password is not the username. You can modify this function based on your requirements.

This function must be created in SYS schema, and you must `connect sys/<password> as sysdba` before running the script.

```
CREATE OR REPLACE FUNCTION verify_function
(username varchar2,
   password varchar2,
   old_password varchar2)
   RETURN boolean IS
   n boolean;
   m integer;
   differ integer;
   isdigit boolean;
   ischar  boolean;
   ispunct boolean;
   digitarray varchar2(20);
   punctarray varchar2(25);
   chararray varchar2(52);

BEGIN
   digitarray:= '0123456789';
   chararray:= 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';
   punctarray:='!"#$%&()''*+,-/:;<=>?_';

   --Check if the password is same as the username
IF password = username THEN
   raise_application_error(-20001, 'Password same as user');
END IF;

   --Check for the minimum length of the password
IF length(password) < 4 THEN
   raise_application_error(-20002, 'Password length less than 4');
END IF;

   --Check if the password is too simple. A dictionary of words may be
   --maintained and a check may be made so as not to allow the words
   --that are too simple for the password.
IF NLS_LOWER(password) IN ('welcome', 'database', 'account', 'user', 'password', 'oracle',
'computer', 'abcd') THEN raise_application_error(-20002, 'Password too simple');
END IF;

   --Check if the password contains at least one letter, one digit and one
   --punctuation mark.
   --1. Check for the digit
   --You may delete 1. and replace with 2. or 3.
isdigit:=FALSE;
m := length(password);
FOR i IN 1..10 LOOP
  FOR j IN 1..m LOOP
    IF substr(password,j,1) = substr(digitarray,i,1) THEN
      isdigit:=TRUE;
```

```
        GOTO findchar;
    END IF;
   END LOOP;
END LOOP;
IF isdigit = FALSE THEN
  raise_application_error(-20003, 'Password should contain at least one
digit, one character and one punctuation');
END IF;
   --2. Check for the character
<<findchar>>
ischar:=FALSE;
FOR i IN 1..length(chararray) LOOP
  FOR j IN 1..m LOOP
    IF substr(password,j,1) = substr(chararray,i,1) THEN
       ischar:=TRUE;
         GOTO findpunct;
       END IF;
    END LOOP;
END LOOP;
IF ischar = FALSE THEN
  raise_application_error(-20003, 'Password should contain at least one digit, one
character and one punctuation');
END IF;
  --3. Check for the punctuation
<<findpunct>>
ispunct:=FALSE;
FOR i IN 1..length(punctarray) LOOP
  FOR j IN 1..m LOOP
    IF substr(password,j,1) = substr(punctarray,i,1) THEN
       ispunct:=TRUE;
         GOTO endsearch;
       END IF;
   END LOOP;
END LOOP;
IF ispunct = FALSE THEN raise_application_error(-20003, 'Password should contain at least
one \ digit, one character and one punctuation');
END IF;

<<endsearch>>

   --Check if the password differs from the previous password by at least 3 letters
IF old_password = '' THEN
  raise_application_error(-20004, 'Old password is null');
END IF;
  --Everything is fine; return TRUE ;
differ := length(old_password) - length(password);

IF abs(differ) < 3 THEN
  IF length(password) < length(old_password) THEN
```

```
    m := length(password);
  ELSE
    m:= length(old_password);
  END IF;
  differ := abs(differ);
  FOR i IN 1..m LOOP
    IF substr(password,i,1) != substr(old_password,i,1) THEN
             differ := differ + 1;
    END IF;
  END LOOP;
  IF differ < 3 THEN
    raise_application_error(-20004, 'Password should differ by at
      least 3 characters');
    END IF;
  END IF;
   --Everything is fine; return TRUE ;
  RETURN(TRUE);
END;
```

Auditing Policy

Security administrators should define a policy for the auditing procedures of each database. You may, for example, decide to have database auditing disabled unless questionable activities are suspected. When auditing is required, the security administrator must decide what level of detail to audit the database; usually, general system auditing is followed by more specific types of auditing after the origins of suspicious activity are determined.

23

Managing Users and Resources

This chapter describes how to control access to an Oracle database, and includes the following topics:

- Session and User Licensing
- User Authentication
- Oracle Users
- Managing Resources with Profiles
- Listing Information About Database Users and Profiles

See Also: For guidelines on establishing security policies for users and profiles, see Chapter 22, "Establishing Security Policies".

Privileges and roles control the access a user has to a database and the schema objects within the database. For information on privileges and roles, see Chapter 24, "Managing User Privileges and Roles".

Session and User Licensing

This section describes aspects of session and user licensing, and includes the following topics:

Oracle helps you ensure that your site complies with its Oracle Server license agreement. If your site is licensed by concurrent usage, you can track and limit the number of sessions concurrently connected to a database. If your site is licensed by named users, you can limit the number of named users created in a database. In either case, you control the licensing facilities, and must enable the facilities and set the appropriate limits.

To use the licensing facility, you need to know which type of licensing agreement your site has, and what the maximum number of sessions or named users is. Your site may use either type of licensing (concurrent usage or named user), but not both.

Note: In a few cases, a site might have an unlimited license, rather than concurrent usage or named user licensing. In these cases only, leave the licensing mechanism disabled, and omit LICENSE_MAX_SESSIONS, LICENSE_SESSIONS_WARNING, and LICENSE_MAX_USERS from the parameter file, or set the value of all three to 0.

Concurrent Usage Licensing

Concurrent usage licensing limits the number of sessions that can be connected simultaneously to the database on the specified computer. You can set a limit on the number of concurrent sessions before you start an instance. In fact, you should have set this limit as part of the initial installation procedure. You can also change the maximum number of concurrent sessions while the database is running.

See Also: For information about the initial installation procedure, see Chapter 2, "Creating an Oracle Database".

Connecting Privileges

After your instance's session limit is reached, only users with RESTRICTED SESSION privilege (usually DBAs) can connect to the database. When a user with RESTRICTED SESSION privileges connects, Oracle sends the user a message indicating that the maximum limit has been reached, and writes a message to the ALERT file. When the maximum is reached, you should connect only to terminate unneeded processes. Do not raise the licensing limits unless you have upgraded your Oracle license agreement.

In addition to setting a maximum concurrent session limit, you can set a warning limit on the number of concurrent sessions. After this limit is reached, additional users can continue to connect (up to the maximum limit); however, Oracle writes an appropriate message to the ALERT file with each connection, and sends each connecting user who has the RESTRICTED SESSION privilege a warning indicating that the maximum is about to be reached.

If a user is connecting with administrator privileges, the limits still apply; however, Oracle enforces the limit after the first statement the user executes.

In addition to enforcing the concurrent usage limits, Oracle tracks the highest number of concurrent sessions for each instance. You can use this "high water mark."

See Also: For information about terminating sessions, see "Terminating Sessions" on page 4-15.

For information about Oracle licensing limit upgrades, see "Viewing Licensing Limits and Current Values" on page 23-6.

Parallel Server Concurrent Usage Limits

For instances running with the Parallel Server, each instance can have its own concurrent usage limit and warning limit. However, the sum of the instances' limits must not exceed the site's concurrent usage license.

WARNING: Sessions that connect to Oracle through multiplexing software or hardware (such as a TP monitor) each contribute individually to the concurrent usage limit. However, the Oracle licensing mechanism cannot distinguish the number of sessions connected this way. If your site uses multiplexing software or hardware, you must consider that and set the maximum concurrent usage limit lower to account for the multiplexed sessions.

See Also: For more information about setting and changing limits in a parallel server environment, see *Oracle8i Parallel Server Concepts and Administration.*

Setting the Maximum Number of Sessions

To set the maximum number of concurrent sessions for an instance, set the parameter LICENSE_MAX_SESSIONS as follows:

```
LICENSE_MAX_SESSIONS = 80
```

If you set this limit, you are not required to set a warning limit (LICENSE_SESSIONS_WARNING). However, using the warning limit makes the maximum limit easier to manage, because it gives you advance notice that your site is nearing maximum use.

Setting the Session Warning Limit

To set the warning limit for an instance, set the parameter LICENSE_SESSIONS_WARNING in the parameter file used to start the instance.

Set the session warning to a value lower than the concurrent usage maximum limit (LICENSE_MAX_SESSIONS).

Changing Concurrent Usage Limits While the Database is Running

To change either the maximum concurrent usage limit or the warning limit while the database is running, use the ALTER SYSTEM command with the appropriate

option. The following statement changes the maximum limit to 100 concurrent sessions:

```
ALTER SYSTEM SET LICENSE_MAX_SESSIONS = 100;
```

The following statement changes both the warning limit and the maximum limit:

```
ALTER SYSTEM
   SET LICENSE_MAX_SESSIONS = 64
   LICENSE_SESSIONS_WARNING = 54;
```

If you change either limit to a value lower than the current number of sessions, the current sessions remain; however, the new limit is enforced for all future connections until the instance is shut down. To change the limit permanently, change the value of the appropriate parameter in the parameter file.

To change the concurrent usage limits while the database is running, you must have the ALTER SYSTEM privilege. Also, to connect to an instance after the instance's maximum limit has been reached, you must have the RESTRICTED SESSION privilege.

WARNING: Do not raise the concurrent usage limits unless you have appropriately upgraded your Oracle Server license. Contact your Oracle representative for more information.

Named User Limits

Named user licensing limits the number of individuals authorized to use Oracle on the specified computer. To enforce this license, you can set a limit on the number of users created in the database before you start an instance. You can also change the maximum number of users while the instance is running, or disable the limit altogether. You cannot create more users after reaching this limit. If you try to do so, Oracle returns an error indicating that the maximum number of users have been created, and writes a message to the ALERT file.

This mechanism operates on the assumption that each person accessing the database has a unique username, and that there are no shared usernames. Do not allow multiple users to connect using the same username.

See Also: For instances running with the Parallel Server, all instances connected to the same database should have the same named user limit. See *Oracle8i Parallel Server Concepts and Administration* for more information.

Setting User Limits

To limit the number of users created in a database, set the LICENSE_MAX_USERS parameter in the database's parameter file. The following example sets the maximum number of users to 200:

```
LICENSE_MAX_USERS = 200
```

If the database contains more than LICENSE_MAX_USERS when you start it, Oracle returns a warning and writes an appropriate message in the ALERT file. You cannot create additional users until the number of users drops below the limit or until you delete users or upgrade your Oracle license.

Changing User Limits

To change the maximum named users limit, use the ALTER SYSTEM command with the LICENSE_MAX_USERS option. The following statement changes the maximum number of defined users to 300:

```
ALTER SYSTEM SET LICENSE_MAX_USERS = 300;
```

If you try to change the limit to a value lower than the current number of users, Oracle returns an error and continues to use the old limit. If you successfully change the limit, the new limit remains in effect until you shut down the instance; to change the limit permanently, change the value of LICENSE_MAX_USERS in the parameter file.

To change the maximum named users limit, you must have the ALTER SYSTEM privilege.

WARNING: Do not raise the named user limit unless you have appropriately upgraded your Oracle license. Contact your Oracle representative for more information.

Viewing Licensing Limits and Current Values

You can see the current limits of all of the license settings, the current number of sessions, and the maximum number of concurrent sessions for the instance by querying the V$LICENSE data dictionary view. You can use this information to determine if you need to upgrade your Oracle license to allow more concurrent sessions or named users:

```
SELECT sessions_max s_max,
   sessions_warning s_warning,
   sessions_current s_current,
```

```
   sessions_highwater s_high,
   users_max
   FROM v$license;

S_MAX    S_WARNING   S_CURRENT    S_HIGH    USERS_MAX
---------------------------------------------------------
100      80          65            82       50
```

In addition, Oracle writes the session high water mark to the database's ALERT file when the database shuts down, so you can check for it there.

To see the current number of named users defined in the database, use the following query:

```
SELECT COUNT(*) FROM dba_users;

COUNT(*)
----------
174
```

User Authentication

This section describes aspects of authenticating users, and includes the following topics:

- Database Authentication
- External Authentication
- Enterprise Authentication

Depending on how you want user identities to be authenticated, there are three ways to define users before they are allowed to create a database session:

1. You can configure Oracle so that it performs both identification and authentication of users. This is called *database authentication*.
2. You can configure Oracle so that it performs only the identification of users (leaving authentication up to the operating system or network service). This is called *external authentication*.
3. You can configure Oracle so that it performs only the identification of users. This is called *enterprise authentication*.

Database Authentication

If you choose database authentication for a user, administration of the user account, password, and authentication of that user is performed entirely by Oracle. To have Oracle authenticate a user, specify a password for the user when you create or alter the user. Users can change their password at any time. Passwords are stored in an encrypted format. Each password must be made up of single-byte characters, even if your database uses a multi-byte character set.

To enhance security when using database authentication, Oracle recommends the use of password management, including account locking, password aging and expiration, password history, and password complexity verification.

The following statement creates a user who is identified and authenticated by Oracle:

```
CREATE USER scott IDENTIFIED BY tiger;
```

See Also: For more information about the CREATE USER and ALTER USER statements, see *Oracle8i SQL Reference.*

For more information about valid passwords, see *Oracle8i SQL Reference.*

For more information about Oracle password management, see Chapter 22, "Establishing Security Policies".

Advantages of Database Authentication

Following are advantages of database authentication:

- User accounts and all authentication are controlled by the database.There is no reliance on anything outside of the database.
- Oracle provides strong password management features to enhance security when using database authentication.
- It is easier to administer small user communities.

External Authentication

When you choose external authentication for a user, the user account is maintained by Oracle, but password administration and user authentication is performed by an external service. This external service can be the operating system or a network service, such as Net8.

With external authentication, your database relies on the underlying operating system or network authentication service to restrict access to database accounts. A

database password is not used for this type of login. If your operating system or network service permits, you can have it authenticate users. If you do so, set the parameter OS_AUTHENT_PREFIX, and use this prefix in Oracle usernames. This parameter defines a prefix that Oracle adds to the beginning of every user's operating system account name. Oracle compares the prefixed username with the Oracle usernames in the database when a user attempts to connect.

For example, assume that OS_AUTHENT_PREFIX is set as follows:

```
OS_AUTHENT_PREFIX=OPS$
```

If a user with an operating system account named "TSMITH" is to connect to an Oracle database and be authenticated by the operating system, Oracle checks that there is a corresponding database user "OPS$TSMITH" and, if so, allows the user to connect. All references to a user authenticated by the operating system must include the prefix, as seen in "OPS$TSMITH".

The default value of this parameter is "OPS$" for backward compatibility with previous versions of Oracle. However, you might prefer to set the prefix value to some other string or a null string (an empty set of double quotes: ""). Using a null string eliminates the addition of any prefix to operating system account names, so that Oracle usernames exactly match operating system usernames.

After you set OS_AUTHENT_PREFIX, it should remain the same for the life of a database. If you change the prefix, any database username that includes the old prefix cannot be used to establish a connection, unless you alter the username to have it use password authentication.

The following command creates a user who is identified by Oracle and authenticated by the operating system or a network service:

```
CREATE USER scott IDENTIFIED EXTERNALLY;
```

Using CREATE USER IDENTIFIED EXTERNALLY, you can create database accounts that must be authenticated via the operating system or network service and cannot be authenticated using a password.

See Also: The text of the OS_AUTHENT_PREFIX parameter is case sensitive on some operating systems. See your operating system-specific Oracle documentation for more information about this initialization parameter.

Operating System Authentication

By default, Oracle only allows operating system authenticated logins over secure connections. Therefore, if you want the operating system to authenticate a user, by default that user cannot connect to the database over Net8. This means the user

cannot connect using a multi-threaded server, since this connection uses Net8. This default restriction prevents a remote user from impersonating another operating system user over a network connection.

If you are not concerned about remote users impersonating another operating system user over a network connection, and you want to use operating system user authentication with network clients, set the parameter REMOTE_OS_AUTHENT (default is FALSE) to TRUE in the database's parameter file. Setting the initialization parameter REMOTE_OS_AUTHENT to TRUE allows the RDBMS to accept the client operating system username received over a non-secure connection and use it for account access. The change will take effect the next time you start the instance and mount the database.

Network Authentication

Network authentication is performed via Net8, which may be configured to use a third party service such as Kerberos. If you are using Net8 as the only external authentication service, the setting of the parameter REMOTE_OS_AUTHENT is irrelevant, since Net8 only allows secure connections.

See Also: For information about network authentication, see *Oracle8i Distributed Database Systems.*

Advantages of External Authentication

Following are advantages of external authentication:

- More choices of authentication mechanism are available, such as smart cards, fingerprints, Kerberos, or the operating system.
- If you are already using some external mechanism for authentication, such as one of those listed above, there may be less administrative overhead to use that mechanism with the database as well.

Enterprise Authentication

If you choose enterprise authentication for a user, the user account is maintained by Oracle, but password administration and user authentication is performed by the Oracle Security Service (OSS). This authentication service can be shared among multiple Oracle database servers and allows user's authentication and authorization information to be managed centrally.

Use the following command to create a user (known as a *global user*) who is identified by Oracle and authenticated by the Oracle Security Service:

```
CREATE USER scott IDENTIFIED GLOBALLY as '<external name>';
```

See Also: For information about the contents of the <EXTERNAL NAME> string, see *Oracle8i Distributed Database Systems.*

Advantages of Enterprise Authentication

Following are advantages of enterprise authentication:

- It is easier to administer large user communities with many databases.
- You can use industry standard public key certificates, giving increased opportunity for interoperability.

See Also: For information about enterprise authentication, see *Oracle8i Distributed Database Systems.*

Oracle Users

Each Oracle database has a list of valid database users. To access a database, a user must run a database application and connect to the database instance using a valid username defined in the database. This section explains how to manage users for a database, and includes the following topics:

- Creating Users
- Altering Users
- Dropping Users

Creating Users

To create a database user, you must have the CREATE USER system privilege. When creating a new user, tablespace quotas can be specified for any tablespace in the database, even if the creator does not have a quota on a specified tablespace. Because it is a powerful privilege, a security administrator is normally the only user who has the CREATE USER system privilege.

You create a user with the SQL statement CREATE USER. Using either option, you can also specify the new user's default and temporary segment tablespaces, tablespace quotas, and profile.

```
CREATE USER OPS$jward
    IDENTIFIED EXTERNALLY
    DEFAULT TABLESPACE data_ts
    TEMPORARY TABLESPACE temp_ts
```

```
    QUOTA 100M ON test_ts
    QUOTA 500K ON data_ts
    PROFILE clerk;
```

See Also: A newly created user cannot connect to the database until granted the CREATE SESSION system privilege; see "Granting System Privileges and Roles" on page 24-9.

Specifying a Name

Within each database a username must be unique with respect to other usernames and roles; a user and role cannot have the same name. Furthermore, each user has an associated schema. Within a schema, each schema object must have a unique name.

Setting a User's Authentication

In the previous CREATE USER statement, the new user is to be authenticated using the operating system. The username includes the default prefix "OPS$." If the OS_AUTHENT_PREFIX parameter is set differently (that is, if it specifies either no prefix or some other prefix), modify the username accordingly, by omitting the prefix or substituting the correct prefix.

Alternatively, you can create a user who is authenticated using the database and a password:

```
CREATE USER jward
   IDENTIFIED BY airplane
   . . . ;
```

In this case, the connecting user must supply the correct password to the database to connect successfully.

User Passwords in Multi-Byte Character Sets In a database that uses a multi-byte character set, passwords must include only single-byte characters. Multi-byte characters are not accepted in passwords.

See Also: For more information about valid passwords, see the *Oracle8i SQL Reference*.

Assigning a Default Tablespace

Each user has a default tablespace. When a user creates a schema object and specifies no tablespace to contain it, Oracle stores the object in the user's default tablespace.

The default setting for every user's default tablespace is the SYSTEM tablespace. If a user does not create objects, this default setting is fine. However, if a user creates any type of object, consider specifically setting the user's default tablespace. You can set a user's default tablespace during user creation, and change it later. Changing the user's default tablespace affects only objects created after the setting is changed.

Consider the following issues when deciding which tablespace to specify:

- Set a user's default tablespace only if the user has the privileges to create objects (such as tables, views, and clusters).
- Set a user's default tablespace to a tablespace for which the user has a quota.
- If possible, set a user's default tablespace to a tablespace other than the SYSTEM tablespace to reduce contention between data dictionary objects and user objects for the same datafiles.

In the previous CREATE USER statement, JWARD's default tablespace is DATA_TS.

Assigning a Temporary Tablespace

Each user also has a temporary tablespace. When a user executes a SQL statement that requires a temporary segment, Oracle stores the segment in the user's temporary tablespace.

If a user's temporary tablespace is not explicitly set, the default is the SYSTEM tablespace. However, setting each user's temporary tablespace reduces file contention among temporary segments and other types of segments. You can set a user's temporary tablespace at user creation, and change it later.

In the previous CREATE USER statement, JWARD's temporary tablespace is TEMP_TS, a tablespace created explicitly to only contain temporary segments.

Assigning Tablespace Quotas

You can assign each user a tablespace quota for any tablespace. Assigning a quota does two things:

- Users with privileges to create certain types of objects can create those objects in the specified tablespace.
- Oracle limits the amount of space that can be allocated for storage of a user's objects within the specified tablespace to the amount of the quota.

By default, a user has no quota on any tablespace in the database. If the user has the privilege to create a schema object, you must assign a quota to allow the user to

create objects. Minimally, assign users a quota for the default tablespace, and additional quotas for other tablespaces in which they will create objects.

You can assign a user either individual quotas for a specific amount of disk space in each tablespace or an unlimited amount of disk space in all tablespaces. Specific quotas prevent a user's objects from consuming too much space in the database.

You can assign a user's tablespace quotas when you create the user, or add or change quotas later. If a new quota is less than the old one, then the following conditions hold true:

- If a user has already exceeded a new tablespace quota, the user's objects in the tablespace cannot be allocated more space until the combined space of these objects falls below the new quota.
- If a user has not exceeded a new tablespace quota, or if the space used by the user's objects in the tablespace falls under a new tablespace quota, the user's objects can be allocated space up to the new quota.

Revoking Tablespace Access You can revoke a user's tablespace access by changing the user's current quota to zero. After a quota of zero is assigned, the user's objects in the revoked tablespace remain, but the objects cannot be allocated any new space.

UNLIMITED TABLESPACE System Privilege To permit a user to use an unlimited amount of any tablespace in the database, grant the user the UNLIMITED TABLESPACE system privilege. This overrides all explicit tablespace quotas for the user. If you later revoke the privilege, explicit quotas again take effect. You can grant this privilege only to users, not to roles.

Before granting the UNLIMITED TABLESPACE system privilege, consider the consequences of doing so:

Advantage

- You can grant a user unlimited access to all tablespaces of a database with one statement.

Disadvantages

- The privilege overrides all explicit tablespace quotas for the user.
- You cannot selectively revoke tablespace access from a user with the UNLIMITED TABLESPACE privilege. You can grant access selectively only after revoking the privilege.

Setting Default Roles

You cannot set a user's default roles in the CREATE USER statement. When you first create a user, the user's default role setting is ALL, which causes all roles subsequently granted to the user to be default roles. Use the ALTER USER command to change the user's default roles.

WARNING: When you create a role (other than a user role), it is granted to you implicitly and added as a default role. You will get an error at login if you have more than MAX_ENABLED_ROLES. You can avoid this error by altering the user's default roles to be less than MAX_ENABLED_ROLES. Thus, you should change the DEFAULT ROLE settings of SYS and SYSTEM before creating user roles.

Altering Users

Users can change their own passwords. However, to change any other option of a user's security domain, you must have the ALTER USER system privilege. Security administrators are normally the only users that have this system privilege, as it allows a modification of *any* user's security domain. This privilege includes the ability to set tablespace quotas for a user on any tablespace in the database, even if the user performing the modification does not have a quota for a specified tablespace.

You can alter a user's security settings with the SQL statement ALTER USER. Changing a user's security settings affects the user's future sessions, not current sessions.

The following statement alters the security settings for user AVYRROS:

```
ALTER USER avyrros
    IDENTIFIED EXTERNALLY
    DEFAULT TABLESPACE data_ts
    TEMPORARY TABLESPACE temp_ts
    QUOTA 100M ON data_ts
    QUOTA 0 ON test_ts
    PROFILE clerk;
```

The ALTER USER statement here changes AVYRROS's security settings as follows:

- Authentication is changed to use AVYRROS's operating system account.
- AVYRROS's default and temporary tablespaces are explicitly set.
- AVYRROS is given a 100M quota for the DATA_TS tablespace.

- AVYRROS's quota on the TEST_TS is revoked.
- AVYRROS is assigned the CLERK profile.

Changing a User's Authentication Mechanism

Most non-DBA users can still change their own passwords with the ALTER USER statement, as follows:

```
ALTER USER andy
   IDENTIFIED BY swordfish;
```

Users can change their own passwords this way, without any special privileges (other than those to connect to the database). Users should be encouraged to change their passwords frequently.

Users must have the ALTER USER privilege to switch between Oracle database authentication, external authentication, and enterprise authentication; usually, only DBAs should have this privilege.

Passwords in Multi-Byte Character Sets In a database that uses a multi-byte character set, passwords must include only single-byte characters. Multi-byte characters are not accepted in passwords.

See Also: For more information about valid passwords, see the *Oracle8i SQL Reference*.

Changing a User's Default Roles

A default role is one that is automatically enabled for a user when the user creates a session. You can assign a user zero or more default roles.

See Also: For more information on changing users' default roles, see Chapter 24, "Managing User Privileges and Roles".

Dropping Users

When a user is dropped, the user and associated schema are removed from the data dictionary and all schema objects contained in the user's schema, if any, are immediately dropped.

> **Note:** If a user's schema and associated objects must remain but the user must be denied access to the database, revoke the CREATE SESSION privilege from the user.

A user that is currently connected to a database cannot be dropped. To drop a connected user, you must first terminate the user's sessions using the SQL statement ALTER SYSTEM with the KILL SESSION clause.

To drop a user and all the user's schema objects (if any), you must have the DROP USER system privilege. Because the DROP USER system privilege is so powerful, a security administrator is typically the only type of user that has this privilege.

You can drop a user from a database using the SQL statement DROP USER.

If the user's schema contains any schema objects, use the CASCADE option to drop the user and all associated objects and foreign keys that depend on the tables of the user successfully. If you do not specify CASCADE and the user's schema contains objects, an error message is returned and the user is not dropped. Before dropping a user whose schema contains objects, thoroughly investigate which objects the user's schema contains and the implications of dropping them before the user is dropped. Pay attention to any unknown cascading effects. For example, if you intend to drop a user who owns a table, check whether any views or procedures depend on that particular table.

```
DROP USER jones CASCADE;
```

See Also: For more information about terminating sessions, see "Terminating Sessions" on page 4-15.

Managing Resources with Profiles

A profile is a named set of resource limits. If resource limits are turned on, Oracle limits database usage and instance resources to whatever is defined in the user's profile. You can assign a profile to each user, and a default profile to all users who do not have specific profiles. For profiles to take effect, resource limits must be turned on for the database as a whole.

This section describes aspects of profile management, and includes the following topics:

- Creating Profiles
- Assigning Profiles
- Altering Profiles
- Using Composite Limits
- Dropping Profiles
- Enabling and Disabling Resource Limits

Creating Profiles

To create a profile, you must have the CREATE PROFILE system privilege. You can create profiles using the SQL statement CREATE PROFILE. At the same time, you can explicitly set particular resource limits.

The following statement creates the profile CLERK:

```
CREATE PROFILE clerk LIMIT
    SESSIONS_PER_USER 2
    CPU_PER_SESSION unlimited
    CPU_PER_CALL 6000
    LOGICAL_READS_PER_SESSION unlimited
    LOGICAL_READS_PER_CALL 100
    IDLE_TIME 30
    CONNECT_TIME 480;
```

All unspecified resource limits for a new profile take the limit set by the DEFAULT profile. You can also specify limits for the DEFAULT profile.

Using the DEFAULT Profile

Each database has a DEFAULT profile, and its limits are used in two cases:

- If a user is not explicitly assigned a profile, then the user conforms to *all* the limits of the DEFAULT profile.
- All unspecified limits of any profile use the corresponding limit of the DEFAULT profile.

Initially, all limits of the DEFAULT profile are set to UNLIMITED. However, to prevent unlimited resource consumption by users of the DEFAULT profile, the security administrator should change the default limits using the ALTER PROFILE statement:

```
ALTER PROFILE default LIMIT
   . . . ;
```

Any user with the ALTER PROFILE system privilege can adjust the limits in the DEFAULT profile. The DEFAULT profile cannot be dropped.

Assigning Profiles

After a profile has been created, you can assign it to database users. Each user can be assigned only one profile at any given time. If a profile is assigned to a user who already has a profile, the new profile assignment overrides the previously assigned

profile. Profile assignments do not affect current sessions. Profiles can be assigned only to users and not to roles or other profiles.

Profiles can be assigned to users using the SQL statements CREATE USER or ALTER USER.

See Also: For more information about assigning a profile to a user, see "Creating Users" on page 23-11 and "Altering Users" on page 23-15.

Altering Profiles

You can alter the resource limit settings of any profile using the SQL statement ALTER PROFILE. To alter a profile, you must have the ALTER PROFILE system privilege.

Any adjusted profile limit overrides the previous setting for that profile limit. By adjusting a limit with a value of DEFAULT, the resource limit reverts to the default limit set for the database. All profiles not adjusted when altering a profile retain the previous settings. Any changes to a profile do not affect current sessions. New profile settings are used only for sessions created after a profile is modified.

The following statement alters the CLERK profile:

```
ALTER PROFILE clerk LIMIT
    CPU_PER_CALL default
    LOGICAL_READS_PER_SESSION 20000;
```

See Also: For information about default profiles, see "Using the DEFAULT Profile" on page 23-18.

Using Composite Limits

You can limit the total resource cost for a session via composite limits. In addition to setting specific resource limits explicitly for a profile, you can set a single composite limit that accounts for resource limits in a profile. You can set a profile's composite limit using the COMPOSITE_LIMIT parameter of the SQL statements CREATE PROFILE or ALTER PROFILE. A composite limit is set via *service units*, which are weighted amounts of each resource.

The following CREATE PROFILE statement is defined using the COMPOSITE_LIMIT parameter:

```
CREATE PROFILE clerk LIMIT
    COMPOSITE_LIMIT 20000
    SESSIONS_PER_USER 2
    CPU_PER_CALL 1000;
```

Notice that both explicit resource limits and a composite limit can exist concurrently for a profile. The limit that is reached first stops the activity in a session. Composite limits allow additional flexibility when limiting the use of system resources.

Determining the Value of the Composite Limit

The correct composite limit depends on the total amount of resource used by an average profile user. As with each specific resource limit, historical information should be gathered to determine the normal range of composite resource usage for a typical profile user.

See Also: For information on how to calculate the composite limit see the *Oracle8i SQL Reference*.

Setting Resource Costs

Each system has its own characteristics; some system resources may be more valuable than others. Oracle enables you to give each system resource a *cost*. Costs weight each system resource at the database level. Costs are only applied to the composite limit of a profile; costs do not apply to set individual resource limits explicitly.

To set resource costs, you must have the ALTER RESOURCE system privilege.

Only certain resources can be given a cost: CPU_PER_ SESSION, LOGICAL_READS_PER_SESSION, CONNECT_TIME, and PRIVATE_SGA. Set costs for a database using the SQL command ALTER RESOURCE COST:

```
ALTER RESOURCE COST
    CPU_PER_SESSION 1
    LOGICAL_READS_PER_SESSION 50;
```

A large cost means that the resource is very expensive, while a small cost means that the resource is not expensive. By default, each resource is initially given a cost of 0. A cost of 0 means that the resource should not be considered in the composite limit (that is, it does not cost anything to use this resource). No resource can be given a cost of NULL.

See Also: For additional information and recommendations on setting resource costs, see your operating system-specific Oracle documentation and the *Oracle8i SQL Reference*.

Dropping Profiles

To drop a profile, you must have the DROP PROFILE system privilege. You can drop a profile using the SQL statement DROP PROFILE. To successfully drop a profile currently assigned to a user, use the CASCADE option.

The following statement drops the profile CLERK, even though it is assigned to a user:

```
DROP PROFILE clerk CASCADE;
```

Any user currently assigned to a profile that is dropped is automatically assigned to the DEFAULT profile. The DEFAULT profile cannot be dropped. Note that when a profile is dropped, the drop does not affect currently active sessions; only sessions created after a profile is dropped abide by any modified profile assignments.

Enabling and Disabling Resource Limits

A profile can be created, assigned to users, altered, and dropped at any time by any authorized database user, but the resource limits set for a profile are enforced only when you enable resource limitation for the associated database. Resource limitation enforcement can be enabled or disabled by two different methods, as described in the next two sections.

To alter the enforcement of resource limitation while the database remains open, you must have the ALTER SYSTEM system privilege.

Enabling and Disabling Resource Limits Before Startup

If a database can be temporarily shut down, resource limitation can be enabled or disabled by the RESOURCE_LIMIT initialization parameter in the database's parameter file. Valid values for the parameter are TRUE (enables enforcement) and FALSE; by default, this parameter's value is set to FALSE. Once the parameter file has been edited, the database instance must be restarted to take effect. Every time an instance is started, the new parameter value enables or disables the enforcement of resource limitation.

Enabling and Disabling Resource Limits While the Database is Open

If a database cannot be temporarily shut down or the resource limitation feature must be altered temporarily, you can enable or disable the enforcement of resource limitation using the SQL command ALTER SYSTEM. After an instance is started, an ALTER SYSTEM statement overrides the value set by the RESOURCE_LIMIT

parameter. For example, the following statement enables the enforcement of resource limitation for a database:

```
ALTER SYSTEM
   SET RESOURCE_LIMIT = TRUE;
```

> **Note:** This does not apply to password resources.

An ALTER SYSTEM statement does not permanently determine the enforcement of resource limitation. If the database is shut down and restarted, the enforcement of resource limits is determined by the value set for the RESOURCE_LIMIT parameter.

Listing Information About Database Users and Profiles

The data dictionary stores information about every user and profile, including the following:

- all users in a database
- each user's default tablespace for tables, clusters, and indexes
- each user's tablespace for temporary segments
- each user's space quotas, if any
- each user's assigned profile and resource limits
- the cost assigned to each applicable system resource
- each current session's memory usage

The following data dictionary views may be of interest when you work with database users and profiles:

- ALL_USERS
- USER_USERS
- DBA_USERS
- USER_TS_QUOTAS
- DBA_TS_QUOTAS
- USER_PASSWORD_LIMITS
- USER_RESOURCE_LIMITS

- DBA_PROFILES
- RESOURCE_COST
- V$SESSION
- V$SESSTAT
- V$STATNAME

See Also: See the *Oracle8i Reference* for detailed information about each view.

Listing Information about Users and Profiles: Examples

The examples in this section assume a database in which the following statements have been executed:

```
CREATE PROFILE clerk LIMIT
    SESSIONS_PER_USER 1Oracle8i SQL Reference.
    IDLE_TIME 30
    CONNECT_TIME 600;

CREATE USER jfee
    IDENTIFIED BY wildcat
    DEFAULT TABLESPACE users
    TEMPORARY TABLESPACE temp_ts
    QUOTA 500K ON users
    PROFILE clerk;

CREATE USER dcranney
    IDENTIFIED BY bedrock
    DEFAULT TABLESPACE users
    TEMPORARY TABLESPACE temp_ts
    QUOTA unlimited ON users;

CREATE USER userscott
     IDENTIFIED BY "scott1"
     PASSWORD_LIFETIME 60
     PASSWORD_GRACE_TIME 10;
```

Listing All Users and Associated Information

The following query lists users and their associated information as defined in the database:

```
SELECT username, profile, account_status from dba_users;
USERNAME        PROFILE         ACCOUNT_STATUS
--------------- --------------- ----------------
SYS             DEFAULT         OPEN
```

```
SYSTEM          DEFAULT         OPEN
BLAKE           DEFAULT         OPEN
SCOTT           DEFAULT         OPEN
ADAMS           DEFAULT         OPEN
JFEE            DEFAULT         OPEN
DCRANNEY        DEFAULT         OPEN
JONES           DEFAULT         OPEN
CLARK           DEFAULT         OPEN
U               DEFAULT         LOCKED
```

All passwords are encrypted to preserve security.

Listing All Tablespace Quotas

The following query lists all tablespace quotas specifically assigned to each user:

```
SELECT * FROM sys.dba_ts_quotas;
TABLESPACE    USERNAME    BYTES     MAX_BYTES    BLOCKS    MAX_BLOCKS
----------    ---------  --------   ----------   -------   ----------
SYSTEM        SYSTEM        0           0          0          0
SYSTEM        JFEE          0       512000         0        250
SYSTEM        DCRANNEY      0          -1          0         -1
```

When specific quotas are assigned, the exact number is indicated in the MAX_BYTES column. Unlimited quotas are indicated by "-1".

Listing All Profiles and Assigned Limits

The following query lists all profiles in the database and associated settings for each limit in each profile:

```
SELECT * FROM sys.dba_profiles
   ORDER BY profile;
PROFILE                    RESOURCE_NAME              RESOURCE    LIMIT

-------------------------  ---------------            ----------  --------------
DEFAULT                    COMPOSITE_LIMIT            KERNEL     UNLIMITED
DEFAULT                    SESSIONS_PER_USER          KERNEL             1
DEFAULT                    CPU_PER_CALL               KERNEL     UNLIMITED
DEFAULT                    LOGICAL_READS_PER_CALL     KERNEL     UNLIMITED
DEFAULT                    CONNECT_TIME               KERNEL            30
DEFAULT                    IDLE_TIME                  KERNEL           600
DEFAULT                    LOGICAL_READS_PER_SESSION  KERNEL     UNLIMITED
DEFAULT                    CPU_PER_SESSION            KERNEL     UNLIMITED
DEFAULT                    PRIVATE_SGA                KERNEL     UNLIMITED
DEFAULT                    FAILED_LOGIN_ATTEMPTS      PASSWORD   UNLIMITED
DEFAULT                    PASSWORD_LIFE_TIME         PASSWORD   UNLIMITED
DEFAULT                    PASSWORD_REUSE_MAX         PASSWORD   UNLIMITED
```

```
DEFAULT                        PASSWORD_LOCK_TIME          PASSWORD   UNLIMITED
DEFAULT                        PASSWORD_GRACE_TIME         PASSWORD   UNLIMITED
DEFAULT                        PASSWORD_VERIFY_FUNCTION    PASSWORD   UNLIMITED
DEFAULT                        PASSWORD_REUSE_TIME         PASSWORD   UNLIMITED
PROF                           COMPOSITE_LIMIT             KERNEL     DEFAULT
PROF                           PRIVATE_SGA                 KERNEL     DEFAULT
PROF                           CONNECT_TIME                KERNEL     DEFAULT
PROF                           IDLE_TIME                   KERNEL     DEFAULT
PROF                           LOGICAL_READS_PER_CALL      KERNEL     DEFAULT
PROF                           LOGICAL_READS_PER_SESSION   KERNEL     DEFAULT
PROF                           SESSIONS_PER_USER           KERNEL     DEFAULT
PROF                           CPU_PER_CALL                KERNEL     DEFAULT
PROF                           CPU_PER_SESSION             KERNEL     DEFAULT
PROF                           FAILED_LOGIN_ATTEMPTS       PASSWORD          5
PROF                           PASSWORD_LIFE_TIME          PASSWORD         60
PROF                           PASSWORD_REUSE_MAX          PASSWORD   UNLIMITED
PROF                           PASSWORD_LOCK_TIME          PASSWORD          1
PROF                           PASSWORD_GRACE_TIME         PASSWORD         10
PROF                           PASSWORD_VERIFY_FUNCTION    PASSWORD   UNLIMITED
PROF                           PASSWORD_REUSE_TIME         PASSWORD         60
32 rows selected.
```

Viewing Memory Use Per User Session

The following query lists all current sessions, showing the Oracle user and current memory use per session:

```
SELECT username, value || 'bytes' "Current session memory"
   FROM v$session sess, v$sesstat stat, v$statname name
WHERE sess.sid = stat.sid
   AND stat.statistic# = name.statistic#
   AND name.name = 'SESSION_MEMORY';
```

The amount of space indicated in "Current session memory" is allocated in the shared pool for each session connected through the multi-threaded server. You can limit the amount of memory allocated per user with the PRIVATE_SGA resource limit.

To see the maximum memory ever allocated to each session since the instance started, replace 'session memory' in the query above with 'max session memory'.

Examples

This section contains examples that use functions described throughout this chapter.

1. The following statement creates the profile prof:

```
CREATE PROFILE prof limit
  FAILED_LOGIN_ATTEMPTS 5
  PASSWORD_LIFE_TIME 60
  PASSWORD_REUSE_MAX 60
  PASSWORD_REUSE_TIME UNLIMITED
  PASSWORD_VERIFY_FUNCTION verify_function
  PASSWORD_LOCK_TIME 1
  PASSWORD_GRACE_TIME 10;
```

2. The following statement creates a user with the same password as the username with profile prof;

```
CREATE USER userscott IDENTIFIED BY userscott PROFILE prof;
ORA-28003: Password verification for the specified password failed
ORA-20001: Password same as user
```

3. The following statement creates user userscott identified by "scott1%" with profile prof;

```
CREATE USER userscott IDENTIFIED BY "scott%" PROFILE prof;
```

4. The following statement changes the user's password to "scott%" again and returns an error:

```
ALTER USER userscott IDENTIFIED BY "scott%";
ORA-28007: The password cannot be reused
```

5. The following statement locks the user account:

```
ALTER USER userscott ACCOUNT LOCK;
```

6. The following statement checks the user account status:

```
SELECT username, user_id, account_status, lock_date
   FROM dba_users
   WHERE username='USERSCOTT';
```

7. The following statement expires the password:

```
ALTER USER userscott PASSWORD EXPIRE;
```

8. The following statement checks the user account status:

```
SELECT username, user_id, account_status, expiry_date
   FROM dba_users
   WHERE username='USERSCOTT';
```

9. The following statement unlocks the user:

```
ALTER USER userscott ACCOUNT UNLOCK;
```

10. The following statement checks the account status:

```
SELECT username, user_id, account_status, expiry_date
   FROM dba_users
   WHERE username='USERSCOTT';
```

24

Managing User Privileges and Roles

This chapter explains how to control the ability to execute system operations and access to schema objects using privileges and roles. The following topics are included:

- Identifying User Privileges
- Managing User Roles
- Granting User Privileges and Roles
- Revoking User Privileges and Roles
- Granting Roles Using the Operating System or Network
- Listing Privilege and Role Information

See Also: For information about controlling access to a database, see Chapter 23.

For suggested general database security policies, see Chapter 22.

Identifying User Privileges

This section describes Oracle user privileges, and includes the following topics:

- System Privileges
- Object Privileges

A user *privilege* is a right to execute a particular type of SQL statement, or a right to access another user's object. Oracle also provides shortcuts for grouping privileges that are commonly granted or revoked together.

System Privileges

There are over 100 distinct system privileges. Each system privilege allows a user to perform a particular database operation or class of database operations.

For security reasons, system privileges do not allow users to access the data dictionary. Hence, users with ANY privileges (such as UPDATE ANY TABLE, SELECT ANY TABLE or CREATE ANY INDEX) cannot access dictionary tables and views that have not been granted to PUBLIC.

> **WARNING: System privileges can be very powerful, and should be cautiously granted to roles and trusted users of the database. Users with the ANY privilege cannot access the data dictionary.**

See Also: For a complete list / description of system privileges, see the *Oracle8i SQL Reference*.

System Privilege Restrictions

The dictionary protection mechanism prevents unauthorized users from accessing dictionary objects.

Access to dictionary objects is restricted to the users SYSDBA and SYSOPER. System privileges providing access to objects in other schemas do *not* give you access to dictionary objects. For example, the SELECT ANY TABLE privilege allows you to access views and tables in other schemas, but does not enable you to select dictionary objects (base tables of dynamic performance views, views, packages, and synonyms).

Also, attempting to connect with the SQL*Plus command `connect SYS/password` results in failure. However, the following two SQL*Plus commands are valid:

```
connect SYS/password as SYSDBA
connect SYS/password as SYSOPER
```

Accessing Frequently Used Dictionary Objects

Users with explicit object privileges and the SYSDBA can access dictionary objects. If, however, you need access to dictionary objects and do not have explicit object privileges, you can be granted the following roles:

- SELECT_CATALOG_ROLE

  Enables users to SELECT all exported catalog views and tables granted to this role. Grant this role to users who must access all exported views and tables in the data dictionary.

- EXECUTE_CATALOG_ROLE

  Provides EXECUTE privilege on exported packages in the dictionary.

- DELETE_CATALOG_ROLE

  Enables users to delete records from the AUD$ table.

These roles enable database administrators to access certain objects in the dictionary while maintaining dictionary security.

Note: SYSDBA should not grant any user the object privileges for nonexported objects in the dictionary; doing so may compromise the integrity of the database.

See Also: For details about any exported table or view, see the *Oracle8i Reference.*

Object Privileges

Each type of object has different privileges associated with it. For a detailed list of objects and associated privileges, see the *Oracle8i SQL Reference.*

Object Privilege Shortcut

The ALL and ALL PRIVILEGES shortcuts grant or revoke all available object privileges for a object. This shortcut is not a privilege; rather, it is a way of granting or revoking all object privileges with one word in GRANT and REVOKE

statements. Note that if all object privileges are granted using the ALL shortcut, individual privileges can still be revoked.

Likewise, all individually granted privileges can be revoked using the ALL shortcut. However, if you REVOKE ALL, and revoking causes integrity constraints to be deleted (because they depend on a REFERENCES privilege that you are revoking), you must include the CASCADE CONSTRAINTS option in the REVOKE statement.

Managing User Roles

This section describes aspects of managing roles, and includes the following topics:

- Creating a Role
- Predefined Roles

A *role* groups several privileges and roles, so that they can be granted to and revoked from users simultaneously. Roles can be enabled and disabled per user.

See Also: For information about roles, see *Oracle8i Concepts.*

Creating a Role

You can create a role using the SQL statement CREATE ROLE.

You must have the CREATE ROLE system privilege to create a role. Typically, only security administrators have this system privilege.

Note: Immediately after creation, a role has no privileges associated with it. To associate privileges with a new role, you must grant privileges or other roles to the new role.

The following statement creates the CLERK role, which is authorized by the database using the password BICENTENNIAL:

```
CREATE ROLE clerk
IDENTIFIED BY bicentennial;
```

Role Names

You must give each role you create a unique name among existing usernames and role names of the database. Roles are not contained in the schema of any user.

Role Names in Multi-Byte Character Sets

In a database that uses a multi-byte character set, Oracle recommends that each role name contain at least one single-byte character. If a role name contains only multi-byte characters, the encrypted role name/password combination is considerably less secure.

Predefined Roles

The roles listed in Table 24–1 are automatically defined for Oracle databases. These roles are provided for backward compatibility to earlier versions of Oracle. You can grant and revoke privileges and roles to these predefined roles, much the way you do with any role you define.

Table 24–1 Predefined Roles

Role Name	Privileges Granted To Role
CONNECT[1]	ALTER SESSION, CREATE CLUSTER, CREATE DATABASE LINK, CREATE SEQUENCE, CREATE SESSION, CREATE SYNONYM, CREATE TABLE, CREATE VIEW
CREATE TYPE [7]	CREATE TYPE, EXECUTE, EXECUTE ANY TYPE
RESOURCE [1,2]	CREATE CLUSTER, CREATE INDEXTYPE, CREATE OPERATOR, CREATE PROCEDURE, CREATE SEQUENCE, CREATE TABLE, CREATE TRIGGER, CREATE TYPE
DBA [1,3,4]	All system privileges WITH ADMIN OPTION
EXP_FULL_DATABASE [5]	SELECT ANY TABLE, BACKUP ANY TABLE, INSERT, DELETE, AND UPDATE ON THE TABLES SYS.INCVID, SYS.INCFIL, AND SYS.INCEXP
IMP_FULL_DATABASE [5]	BECOME USER
DELETE_CATALOG_ROLE [6]	DELETE privileges on all dictionary packages for this role.
EXECUTE_CATALOG_ROLE [6]	EXECUTE privilege on all dictionary packages for this role.
SELECT_CATALOG_ROLE [6]	SELECT privilege on all catalog tables and views for this role.

Table 24–1 Predefined Roles (Cont.)

Role Name	Privileges Granted To Role
RECOVERY_CATALOG_OWNER[8]	DROP ROLE RECOVERY_CATALOG_OWNER, CREATE ROLE RECOVERY_CATALOG_OWNER,, CREATE TRIGGER, CREATE PROCEDURE TO RECOVERY_CATALOG_OWNER
HS_ADMIN_ROLE[9]	HS_EXTERNAL_OBJECT, HS_EXTERNAL_USER
AQ_USER_ROLE[10]	
AQ_ADMINISTRATOR_ROLE[10]	
SNMPAGENT[11]	

[1]Created by SQL.BSQ. For backward compatibility. Not recommended for use.

[2]Grantees of the RESOURCE role also receive the UNLIMITED TABLESPACE system privilege as an explicit grant (not as part of the RESOURCE role). For backward compatibility. Not recommended for use.

[3]Grantees of the DBA role also receive the UNLIMITED TABLESPACE system privilege with the ADMIN option as an explicit grant (not as part of the DBA role). Therefore when the DBA role is revoked, any explicit grant of UNLIMITED TABLESPACE is also revoked.

[4]Also includes the EXP_FULL_DATABASE and IMP_FULL_DATABASE roles if CATEXP.SQL has been run.

[5]Created by CATEXP.SQL.

[6]These roles must be granted to users who do not have the DBA role, but require access to the views and tables in the data dictionary.

[7]The CREATE TYPE command is only available if the Oracle objects option is installed on your database server.

[8]Created by CAT.SQL.

[9]Created by CATQUEUE.SQL.

[10]Only granted when you have the Advanced Queuing option.

[11]Only granted when you have the Intelligent Agents option.

Role Authorization

A database role can optionally require authorization when a user attempts to enable the role. Role authorization can be maintained by the database (using passwords), by the operating system, or by a network service.

To alter the authorization method for a role, you must have the ALTER ANY ROLE system privilege or have been granted the role with the ADMIN OPTION.

See Also: For more information about network roles, see *Oracle8i Distributed Database Systems.*

Role Authorization by the Database

The use of a role can be protected by an associated password. If you are granted a role protected by a password, you can enable or disable the role only by supplying the proper password for the role in a SET ROLE statement.

Note: In a database that uses a multi-byte character set, passwords for roles must include only single-byte characters. Multi-byte characters are not accepted in passwords.

See Also: For more information about valid passwords, see the *Oracle8i Reference*.

Role Authorization by the Operating System

The following statement creates a role named ACCTS_REC and requires that the operating system authorize its use:

```
CREATE ROLE role IDENTIFIED EXTERNALLY;
```

Role authentication via the operating system is useful only when the operating system is able to dynamically link operating system privileges with applications. When a user starts an application, the operating system grants an operating system privilege to the user. The granted operating system privilege corresponds to the role associated with the application. At this point, the application can enable the application role. When the application is terminated, the previously granted operating system privilege is revoked from the user's operating system account.

If a role is authorized by the operating system, you must configure information for each user at the operating system level. This operation is operating system dependent.

If roles are granted by the operating system, you do not need to have the operating system authorize them also; this is redundant.

See Also: For more information about roles granted by the operating system, see "Granting Roles Using the Operating System or Network" on page 24-16.

Role Authorization and Network Clients

If users connect to the database over Net8, by default their roles cannot be authenticated by the operating system. This includes connections through a multi-threaded server, as this connection requires Net8. This restriction is the default

because a remote user could impersonate another operating system user over a network connection.

If you are not concerned with this security risk and want to use operating system role authentication for network clients, set the parameter REMOTE_OS_ROLES in the database's parameter file to TRUE. The change will take effect the next time you start the instance and mount the database. (The parameter is FALSE by default.)

Withholding Authorization

A role can also be created without authorization. If a role is created without any protection, the role can be enabled or disabled by any grantee.

Changing a Role's Authorization

You can set and change the authorization method for a role using the SQL statement ALTER ROLE.

The following statement alters the CLERK role to be authorized externally:

```
ALTER ROLE clerk
IDENTIFIED EXTERNALLY;
```

Changing a User's Default Roles

A user's list of default roles can be set and altered using the SQL statement ALTER USER.

See Also: See "Altering Users" on page 23-15 for more information about these options.

Using the MAX_ENABLED_ROLES Parameter A user can enable as many roles as specified by the initialization parameter MAX_ENABLED_ROLES. All indirectly granted roles enabled as a result of enabling a primary role are included in this count. The database administrator can alter this limitation by modifying the value for this parameter. Higher values permit each user session to have more concurrently enabled roles. However, the larger the value for this parameter, the more memory space is required on behalf of each user session; this is because the PGA size is affected for each user session, and requires 4 bytes per role. Determine the highest number of roles that will be concurrently enabled by any one user and use this value for the MAX_ENABLED_ROLES parameter.

Dropping Roles

In some cases, it may be appropriate to drop a role from the database. The security domains of all users and roles granted a dropped role are immediately changed to

reflect the absence of the dropped role's privileges. All indirectly granted roles of the dropped role are also removed from affected security domains. Dropping a role automatically removes the role from all users' default role lists.

Because the creation of objects is not dependent on the privileges received via a role, tables and other objects are not dropped when a role is dropped.

To drop a role, you must have the DROP ANY ROLE system privilege or have been granted the role with the ADMIN OPTION.

You can drop a role using the SQL statement DROP ROLE.

The following statement drops the role CLERK:

```
DROP ROLE clerk;
```

Granting User Privileges and Roles

This section describes aspects of granting privileges and roles, and includes the following topics:

- Granting System Privileges and Roles
- Granting Object Privileges and Roles
- Granting Privileges on Columns

Granting System Privileges and Roles

You can grant system privileges and roles to other roles and users using the SQL statement GRANT.

To grant a system privilege or role, you must have the ADMIN OPTION for all system privileges and roles being granted. Also, any user with the GRANT ANY ROLE system privilege can grant any role in a database.

The following statement grants the system privilege and the ACCTS_PAY role to the user JWARD:

```
GRANT create session, accts_pay
TO jward;
```

Note: Object privileges *cannot* be granted along with system privileges and roles in the same GRANT statement.

The ADMIN Option

When a user creates a role, the role is automatically granted to the creator with the ADMIN OPTION. A grantee with the ADMIN option has several expanded capabilities:

- The grantee can grant or revoke the system privilege or role to or from *any* user or other role in the database. (Users cannot revoke a role from themselves.)
- The grantee can further grant the system privilege or role with the ADMIN OPTION.
- The grantee of a role can alter or drop the role.

In the following statement, the security administrator grants the NEW_DBA role to MICHAEL:

```
GRANT new_dba TO michael WITH ADMIN OPTION;
```

The user MICHAEL can not only use all of the privileges implicit in the NEW_DBA role, but can grant, revoke, or drop the NEW_DBA role as deemed necessary. Because of these powerful capabilities, exercise caution when granting system privileges or roles with the ADMIN OPTION. Such privileges are usually reserved for a security administrator and rarely granted to other administrators or users of the system.

Granting Object Privileges and Roles

You can grant object privileges to roles and users using the SQL command GRANT.

To grant an object privilege, you must fulfill one of the following conditions:

- You own the object specified.
- You have been granted the object privileges being granted with the GRANT OPTION.

The following statement grants the SELECT, INSERT, and DELETE object privileges for all columns of the EMP table to the users JFEE and TSMITH:

```
GRANT select, insert, delete ON emp TO jfee, tsmith;
```

To grant the INSERT object privilege for only the ENAME and JOB columns of the EMP table to the users JFEE and TSMITH, issue the following statement:

```
GRANT insert(ename, job) ON emp TO jfee, tsmith;
```

To grant all object privileges on the SALARY view to the user JFEE, use the ALL shortcut, as shown in the following example:

```
GRANT ALL ON salary TO jfee;
```

Note: System privileges and roles cannot be granted along with object privileges in the same GRANT statement.

The GRANT OPTION

The user whose schema contains an object is automatically granted all associated object privileges with the GRANT OPTION. This special privilege allows the grantee several expanded privileges:

- The grantee can grant the object privilege to any users in the database, with or without the GRANT OPTION, or to any role in the database.
- If the grantee receives object privileges for a table with the GRANT OPTION and the grantee has the CREATE VIEW or CREATE ANY VIEW system privilege, the grantee can create views on the table and grant the corresponding privileges on the view to any user or role in the database.

The GRANT OPTION is not valid when granting an object privilege to a role. Oracle prevents the propagation of object privileges via roles so that grantees of a role cannot propagate object privileges received by means of roles.

Granting Privileges on Columns

You can grant INSERT, UPDATE, or REFERENCES privileges on individual columns in a table.

> **WARNING: Before granting a column-specific INSERT privilege, determine if the table contains any columns on which NOT NULL constraints are defined. Granting selective insert capability without including the NOT NULL columns prevents the user from inserting any rows into the table. To avoid this situation, make sure that each NOT NULL column is either insertable or has a non-NULL default value. Otherwise, the grantee will not be able to insert rows into the table and will receive an error.**

Grant INSERT privilege on the ACCT_NO column of the ACCOUNTS table to SCOTT:

```
GRANT INSERT (acct_no)
ON accounts TO scott;
```

Revoking User Privileges and Roles

This section describes aspects of revoking user privileges and roles, and includes the following topics:

- Revoking System Privileges and Roles
- Revoking Object Privileges and Roles

Revoking System Privileges and Roles

You can revoke system privileges and / or roles using the SQL statement REVOKE.

Any user with the ADMIN OPTION for a system privilege or role can revoke the privilege or role from any other database user or role The revoker does not have to be the user that originally granted the privilege or role. Also, users with the GRANT ANY ROLE can revoke *any* role.

The following statement revokes the CREATE TABLE system privilege and the ACCTS_REC role from TSMITH:

```
REVOKE create table, accts_rec FROM tsmith;
```

Note: The ADMIN OPTION for a system privilege or role cannot be selectively revoked. The privilege or role must be revoked and then the privilege or role re-granted without the ADMIN OPTION.

Revoking Object Privileges and Roles

You can revoke object privileges using the SQL command REVOKE.

To revoke an object privilege, the revoker must be the original grantor of the object privilege being revoked.

For example, assuming you are the original grantor, to revoke the SELECT and INSERT privileges on the EMP table from the users JFEE and TSMITH, you would issue the following statement:

```
REVOKE select, insert ON emp
FROM jfee, tsmith;
```

The following statement revokes all privileges (which were originally granted to the role HUMAN_RESOURCE) from the table DEPT:

```
REVOKE ALL ON dept FROM human_resources;
```

Note: This statement above would only revoke the privileges that the grantor authorized, not the grants made by other users. The GRANT OPTION for an object privilege cannot be selectively revoked. The object privilege must be revoked and then re-granted without the GRANT OPTION. Users cannot revoke object privileges from themselves.

Revoking Column-Selective Object Privileges

Although users can grant column-selective INSERT, UPDATE, and REFERENCES privileges for tables and views, they cannot selectively revoke column specific privileges with a similar REVOKE statement. Instead, the grantor must first revoke the object privilege for all columns of a table or view, and then selectively re-grant the column-specific privileges that should remain.

For example, assume that role HUMAN_RESOURCES has been granted the UPDATE privilege on the DEPTNO and DNAME columns of the table DEPT. To

revoke the UPDATE privilege on just the DEPTNO column, you would issue the following two statements:

```
REVOKE UPDATE ON dept FROM human_resources;
GRANT UPDATE (dname) ON dept TO human_resources;
```

The REVOKE statement revokes UPDATE privilege on all columns of the DEPT table from the role HUMAN_RESOURCES. The GRANT statement re-grants UPDATE privilege on the DNAME column to the role HUMAN_RESOURCES.

Revoking the REFERENCES Object Privilege

If the grantee of the REFERENCES object privilege has used the privilege to create a foreign key constraint (that currently exists), the grantor can revoke the privilege only by specifying the CASCADE CONSTRAINTS option in the REVOKE statement:

```
REVOKE REFERENCES ON dept FROM jward CASCADE CONSTRAINTS;
```

Any foreign key constraints currently defined that use the revoked REFERENCES privilege are dropped when the CASCADE CONSTRAINTS options is specified.

Effects of Revoking Privileges

Depending on the type of privilege, there may be cascading effects when a privilege is revoked.

System Privileges

There are no cascading effects when revoking a system privilege related to DDL operations, regardless of whether the privilege was granted with or without the ADMIN OPTION. For example, assume the following:

1. The security administrator grants the CREATE TABLE system privilege to JFEE with the ADMIN OPTION.
2. JFEE creates a table.
3. JFEE grants the CREATE TABLE system privilege to TSMITH.
4. TSMITH creates a table.
5. The security administrator revokes the CREATE TABLE system privilege from JFEE.
6. JFEE's table continues to exist. TSMITH still has the table and the CREATE TABLE system privilege.

Cascading effects can be observed when revoking a system privilege related to a DML operation. For example, if SELECT ANY TABLE is granted to a user, and that user has created any procedures, all procedures contained in the user's schema must be re-authorized before they can be used again.

Object Privileges

Revoking an object privilege may have cascading effects that should be investigated before issuing a REVOKE statement.

- Object definitions that depend on a DML object privilege can be affected if the DML object privilege is revoked. For example, assume the procedure body of the TEST procedure includes a SQL statement that queries data from the EMP table. If the SELECT privilege on the EMP table is revoked from the owner of the TEST procedure, the procedure can no longer be executed successfully.
- Object definitions that require the ALTER and INDEX DDL object privileges are not affected if the ALTER or INDEX object privilege is revoked. For example, if the INDEX privilege is revoked from a user that created an index on someone else's table, the index continues to exist after the privilege is revoked.
- When a REFERENCES privilege for a table is revoked from a user, any foreign key integrity constraints defined by the user that require the dropped REFERENCES privilege are automatically dropped. For example, assume that the user JWARD is granted the REFERENCES privilege for the DEPTNO column of the DEPT table and creates a foreign key on the DEPTNO column in the EMP table that references the DEPTNO column. If the REFERENCES privilege on the DEPTNO column of the DEPT table is revoked, the foreign key constraint on the DEPTNO column of the EMP table is dropped in the same operation.
- The object privilege grants propagated using the GRANT OPTION are revoked if a grantor's object privilege is revoked. For example, assume that USER1 is granted the SELECT object privilege with the GRANT OPTION, and grants the SELECT privilege on EMP to USER2. Subsequently, the SELECT privilege is revoked from USER1. This revoke is cascaded to USER2 as well. Any objects that depended on USER1's and USER2's revoked SELECT privilege can also be affected, as described in previous bullet items.

Granting to and Revoking from the User Group PUBLIC

Privileges and roles can also be granted to and revoked from the user group PUBLIC. Because PUBLIC is accessible to every database user, all privileges and roles granted to PUBLIC are accessible to every database user.

Security administrators and database users should grant a privilege or role to PUBLIC only if every database user requires the privilege or role. This recommendation reinforces the general rule that at any given time, each database user should only have the privileges required to accomplish the group's current tasks successfully.

Revoking a privilege from PUBLIC can cause significant cascading effects. If any privilege related to a DML operation is revoked from PUBLIC (for example, SELECT ANY TABLE, UPDATE ON emp), all procedures in the database, including functions and packages, must be *reauthorized* before they can be used again. Therefore, exercise caution when granting DML-related privileges to PUBLIC.

See Also: For more information about object dependencies, see "Managing Object Dependencies" on page 20-23.

When Do Grants and Revokes Take Effect?

Depending on what is granted or revoked, a grant or revoke takes effect at different times:

- All grants / revokes of system and object privileges to anything (users, roles, and PUBLIC) are immediately observed.
- All grants / revokes of roles to anything (users, other roles, PUBLIC) are only observed when a current user session issues a SET ROLE statement to re-enable the role after the grant / revoke, or when a new user session is created after the grant / revoke.

Granting Roles Using the Operating System or Network

This section describes aspects of granting roles via your operating system or network, and includes the following topics:

- Using Operating System Role Identification
- Using Operating System Role Management
- Granting and Revoking Roles When OS_ROLES=TRUE
- Enabling and Disabling Roles When OS_ROLES=TRUE
- Using Network Connections with Operating System Role Management

Instead of a security administrator explicitly granting and revoking database roles to and from users using GRANT and REVOKE statements, the operating system that operates Oracle can grant roles to users at connect time. Roles can be

administered using the operating system and passed to Oracle when a user creates a session. As part of this mechanism, each user's default roles and the roles granted to a user with the ADMIN OPTION can be identified. Even if the operating system is used to authorize users for roles, all roles must be created in the database and privileges assigned to the role with GRANT statements.

Roles can also be granted through a network service. For information about network roles, see *Oracle8i Distributed Database Systems.*

The advantage of using the operating system to identify a user's database roles is that privilege management for an Oracle database can be externalized. The security facilities offered by the operating system control a user's privileges. This option may offer advantages of centralizing security for a number of system activities. For example, MVS Oracle administrators may want RACF groups to identify a database user's roles, UNIX Oracle administrators may want UNIX groups to identify a database user's roles, or VMS Oracle administrators may want to use rights identifiers to identify a database user's roles.

The main disadvantage of using the operating system to identify a user's database roles is that privilege management can only be performed at the role level. Individual privileges cannot be granted using the operating system, but can still be granted inside the database using GRANT statements.

A secondary disadvantage of using this feature is that by default users cannot connect to the database through the multi-threaded server, or any other network connection, if the operating system is managing roles. However, you can change this default; see "Using Network Connections with Operating System Role Management" on page 24-19.

See Also: The features described in this section are available only on some operating systems. This information is operating system-dependent; see your operating system-specific Oracle documentation.

Using Operating System Role Identification

To operate a database so that it uses the operating system to identify each user's database roles when a session is created, set the initialization parameter OS_ROLES to TRUE (and restart the instance, if it is currently running). When a user attempts to create a session with the database, Oracle initializes the user's security domain using the database roles identified by the operating system.

To identify database roles for a user, each Oracle user's operating system account must have operating system identifiers (these may be called groups, rights identifiers, or other similar names) that indicate which database roles are to be

available for the user. Role specification can also indicate which roles are the default roles of a user and which roles are available with the ADMIN OPTION. No matter which operating system is used, the role specification at the operating system level follows the format:

```
ORA_<ID>_<ROLE>[_[D][A]]
where:
```

ID
The definition of ID varies on different operating systems. For example, on VMS, ID is the instance identifier of the database; on MVS, it is the machine type; on UNIX, it is the system ID.

D
This optional character indicates that this role is to be a default role of the database user.

A
This optional character indicates that this role is to be granted to the user with the ADMIN OPTION. This allows the user to grant the role to other roles only. (Roles cannot be granted to users if the operating system is used to manage roles.)

> **Note:** If either the D or A characters are specified, they must be preceded by an underscore.

For example, an operating system account might have the following roles identified in its profile:

```
ORA_PAYROLL_ROLE1
ORA_PAYROLL_ROLE2_A
ORA_PAYROLL_ROLE3_D
ORA_PAYROLL_ROLE4_DA
```

When the corresponding user connects to the PAYROLL instance of Oracle, ROLE3 and ROLE4 are defaults, while ROLE2 and ROLE4 are available with the ADMIN OPTION.

Using Operating System Role Management

When you use operating system managed roles, it is important to note that database roles are being granted to an operating system user. Any database user to which the OS user is able to connect will have the authorized database roles enabled. For this reason, you should consider defining all Oracle users as IDENTIFIED

EXTERNALLY if you are using OS_ROLES = TRUE, so that the database accounts are tied to the OS account that was granted privileges.

Granting and Revoking Roles When OS_ROLES=TRUE

If OS_ROLES is set to TRUE, the operating system completely manages the grants and revokes of roles *to users*. Any previous grants of roles to users via GRANT statements do not apply; however, they are still listed in the data dictionary. Only the role grants made at the operating system level to users apply. Users can still grant privileges to roles and users.

Note: If the operating system grants a role to a user with the ADMIN OPTION, the user can grant the role only to other roles.

Enabling and Disabling Roles When OS_ROLES=TRUE

If OS_ROLES is set to TRUE, any role granted by the operating system can be dynamically enabled using the SET ROLE command. If the role was defined to require a password or operating system authorization, that still applies. However, any role not identified in a user's operating system account cannot be specified in a SET ROLE statement, even if a role has been granted using a GRANT statement when OS_ROLES = FALSE. (If you specify such a role, Oracle ignores it.)

When OS_ROLES = TRUE, a user can enable as many roles as specified by the parameter MAX_ENABLED_ROLES.

Using Network Connections with Operating System Role Management

If you want to have the operating system manage roles, by default users cannot connect to the database through the multi-threaded server. This restriction is the default because a remote user could impersonate another operating system user over a non-secure connection.

If you are not concerned with this security risk and want to use operating system role management with the multi-threaded server, or any other network connection, set the parameter REMOTE_OS_ROLES in the database's parameter file to TRUE. The change will take effect the next time you start the instance and mount the database. (The default setting of this parameter is FALSE.)

Listing Privilege and Role Information

To list the grants made for objects, a user can query the following data dictionary views:

- ALL_COL_PRIVS, USER_COL_PRIVS, DBA_COL_PRIVS
- ALL_COL_PRIVS_MADE, USER_COL_PRIVS_MADE
- ALL_COL_PRIVS_RECD, USER_COL_PRIVS_RECD
- ALL_TAB_PRIVS, USER_TAB_PRIVS, DBA_TAB_PRIVS
- ALL_TAB_PRIVS_MADE, USER_TAB_PRIVS_MADE
- ALL_TAB_PRIVS_RECD, USER_TAB_PRIVS_RECD
- DBA_ROLES
- USER_ROLE_PRIVS, DBA_ROLE_PRIVS
- USER_SYS_PRIVS, DBA_SYS_PRIVS
- COLUMN_PRIVILEGES
- ROLE_ROLE_PRIVS, ROLE_SYS_PRIVS, ROLE_TAB_PRIVS
- SESSION_PRIVS, SESSION_ROLES

See Also: See the *Oracle8i Reference* for a detailed description of these data dictionary views.

Listing Privilege and Role Information: Examples

For the following examples, assume the following statements are issued:

```
CREATE ROLE security_admin IDENTIFIED BY honcho;

GRANT create profile, alter profile, drop profile,
    create role, drop any role, grant any role, audit any,
    audit system, create user, become user, alter user, drop user
    TO security_admin WITH ADMIN OPTION;

GRANT SELECT, DELETE ON sys.aud$ TO security_admin;

GRANT security_admin, create session TO swilliams;

GRANT security_admin TO system_administrator;

GRANT create session TO jward;
```

```
GRANT SELECT, DELETE ON emp TO jward;

GRANT INSERT (ename, job) ON emp TO swilliams, jward;
```

Listing All System Privilege Grants

The following query indicates all system privilege grants made to roles and users:

```
SELECT * FROM sys.dba_sys_privs;

GRANTEE            PRIVILEGE                        ADM
--------------     ---------------------------------  ---
SECURITY_ADMIN     ALTER PROFILE                    YES
SECURITY_ADMIN     ALTER USER                       YES
SECURITY_ADMIN     AUDIT ANY                        YES
SECURITY_ADMIN     AUDIT SYSTEM                     YES
SECURITY_ADMIN     BECOME USER                      YES
SECURITY_ADMIN     CREATE PROFILE                   YES
SECURITY_ADMIN     CREATE ROLE                      YES
SECURITY_ADMIN     CREATE USER                      YES
SECURITY_ADMIN     DROP ANY ROLE                    YES
SECURITY_ADMIN     DROP PROFILE                     YES
SECURITY_ADMIN     DROP USER                        YES
SECURITY_ADMIN     GRANT ANY ROLE                   YES
SWILLIAMS          CREATE SESSION                    NO
JWARD              CREATE SESSION                    NO
```

Listing All Role Grants

The following query returns all the roles granted to users and other roles:

```
SELECT * FROM sys.dba_role_privs;

GRANTEE            GRANTED_ROLE                       ADM
------------------ ---------------------------------- ---
SWILLIAMS          SECURITY_ADMIN                     NO
```

Listing Object Privileges Granted to a User

The following query returns all object privileges (not including column-specific privileges) granted to the specified user:

```
SELECT table_name, privilege, grantable FROM sys.dba_tab_privs
    WHERE grantee = 'JWARD';

TABLE_NAME   PRIVILEGE    GRANTABLE
-----------  ------------ ----------
EMP          SELECT       NO
```

```
EMP           DELETE       NO
```

To list all the column-specific privileges that have been granted, use the following query:

```
SELECT grantee, table_name, column_name, privilege
    FROM sys.dba_col_privs;

GRANTEE      TABLE_NAME     COLUMN_NAME      PRIVILEGE
-----------  ------------   -------------     --------------
SWILLIAMS    EMP            ENAME            INSERT
SWILLIAMS    EMP            JOB              INSERT
JWARD        EMP            NAME             INSERT
JWARD        EMP            JOB              INSERT
```

Listing the Current Privilege Domain of Your Session

The following query lists all roles currently enabled for the issuer:

```
SELECT * FROM session_roles;
```

If SWILLIAMS has enabled the SECURITY_ADMIN role and issues this query, Oracle returns the following information:

```
ROLE
------------------------------
SECURITY_ADMIN
```

The following query lists all system privileges currently available in the issuer's security domain, both from explicit privilege grants and from enabled roles:

```
SELECT * FROM session_privs;
```

If SWILLIAMS has the SECURITY_ADMIN role enabled and issues this query, Oracle returns the following results:

```
PRIVILEGE
----------------------------------------
AUDIT SYSTEM
CREATE SESSION
CREATE USER
BECOME USER
ALTER USER
DROP USER
CREATE ROLE
DROP ANY ROLE
GRANT ANY ROLE
AUDIT ANY
CREATE PROFILE
ALTER PROFILE
DROP PROFILE
```

If the SECURITY_ADMIN role is disabled for SWILLIAMS, the first query would have returned no rows, while the second query would only return a row for the CREATE SESSION privilege grant.

Listing Roles of the Database

The DBA_ROLES data dictionary view can be used to list all roles of a database and the authentication used for each role. For example, the following query lists all the roles in the database:

```
SELECT * FROM sys.dba_roles;
```

```
ROLE                 PASSWORD
----------------     --------
CONNECT              NO
RESOURCE             NO
DBA                  NO
SECURITY_ADMIN       YES
```

Listing Information About the Privilege Domains of Roles

The ROLE_ROLE_PRIVS, ROLE_SYS_PRIVS, and ROLE_TAB_PRIVS data dictionary views contain information on the privilege domains of roles.

For example, the following query lists all the roles granted to the SYSTEM_ADMIN role:

```
SELECT granted_role, admin_option
   FROM role_role_privs
   WHERE role = 'SYSTEM_ADMIN';
GRANTED_ROLE              ADM
----------------          ----
SECURITY_ADMIN            NO
```

The following query lists all the system privileges granted to the SECURITY_ADMIN role:

```
SELECT * FROM role_sys_privs WHERE role = 'SECURITY_ADMIN';

ROLE                    PRIVILEGE                      ADM
----------------------- ------------------------------ ---
SECURITY_ADMIN           ALTER PROFILE                 YES
SECURITY_ADMIN           ALTER USER                    YES
SECURITY_ADMIN           AUDIT ANY                     YES
SECURITY_ADMIN           AUDIT SYSTEM                  YES
SECURITY_ADMIN           BECOME USER                   YES
SECURITY_ADMIN           CREATE PROFILE                YES
SECURITY_ADMIN           CREATE ROLE                   YES
SECURITY_ADMIN           CREATE USER                   YES
SECURITY_ADMIN           DROP ANY ROLE                 YES
SECURITY_ADMIN           DROP PROFILE                  YES
SECURITY_ADMIN           DROP USER                     YES
SECURITY_ADMIN           GRANT ANY ROLE                YES
```

The following query lists all the object privileges granted to the SECURITY_ADMIN role:

```
SELECT table_name, privilege FROM role_tab_privs
    WHERE role = 'SECURITY_ADMIN';

TABLE_NAME                     PRIVILEGE
-----------------------------  ----------------
AUD$                           DELETE
AUD$                           SELECT
```

25

Auditing Database Use

This chapter describes how to use the Oracle auditing facilities, and includes the following topics:

- Guidelines for Auditing
- Creating and Deleting the Database Audit Trail Views
- Managing Audit Trail Information
- Viewing Database Audit Trail Information
- Auditing Through Database Triggers

Guidelines for Auditing

This section describes guidelines for auditing and includes the following topics:

- Audit via the Database or Operating System
- Keep Audited Information Manageable

Audit via the Database or Operating System

The data dictionary of every database has a table named SYS.AUD$, commonly referred to as the database *audit trail*.

Either the database or operating system audit trail can store all audit records generated as the result of statement, privilege, or object auditing.

Your operating system may or may not support database auditing to the operating system audit trail. If this option is available, consider the advantages and disadvantages of using either the database or operating system auditing trail to store database audit records.

Using the database audit trail offers the following advantages:

- You can view selected portions of the audit trail with the predefined audit trail views of the data dictionary.
- You can use Oracle tools (such as Oracle Reports) to generate audit reports.

Alternatively, your operating system audit trail may allow you to consolidate audit records from multiple sources including Oracle and other applications. Therefore, examining system activity might be more efficient because all audit records are in one place.

See Also: Your operating system may also contain an audit trail that stores audit records generated by the operating system auditing facility. However, this facility is operating system-dependent. See your operating system-specific Oracle documentation.

Keep Audited Information Manageable

Although auditing is relatively inexpensive, limit the number of audited events as much as possible. This will minimize the performance impact on the execution of statements that are audited, and minimize the size of the audit trail.

Use the following general guidelines when devising an auditing strategy:

- Evaluate your purpose for auditing.

After you have a clear understanding of the reasons for auditing, you can devise an appropriate auditing strategy and avoid unnecessary auditing.

For example, suppose you are auditing to investigate suspicious database activity. This information by itself is not specific enough. What types of suspicious database activity do you suspect or have you noticed? A more focused auditing purpose might be to audit unauthorized deletions from arbitrary tables in the database. This purpose narrows the type of action being audited and the type of object being affected by the suspicious activity.

- Audit knowledgeably.

  Audit the minimum number of statements, users, or objects required to get the targeted information. This prevents unnecessary audit information from cluttering the meaningful information and consuming valuable space in the SYSTEM tablespace. Balance your need to gather sufficient security information with your ability to store and process it.

  For example, if you are auditing to gather information about database activity, determine exactly what types of activities you are tracking, audit only the activities of interest, and audit only for the amount of time necessary to gather the information you desire. Do not audit objects if you are only interested in each session's logical I/O information.

Auditing Suspicious Database Activity

When you audit to monitor suspicious database activity, use the following guidelines:

- Audit generally, then specifically.

  When starting to audit for suspicious database activity, it is common that not much information is available to target specific users or schema objects. Therefore, audit options must be set more generally at first. Once preliminary audit information is recorded and analyzed, the general audit options should be turned off and more specific audit options enabled. This process should continue until enough evidence is gathered to make concrete conclusions about the origin of the suspicious database activity.

- Protect the audit trail.

  When auditing for suspicious database activity, protect the audit trail so that audit information cannot be added, changed, or deleted without being audited.

See Also: For more information about the audit trail, see "Protecting the Audit Trail" on page 25-16.

Auditing Normal Database Activity

When your purpose for auditing is to gather historical information about particular database activities, use the following guidelines:

- Audit only pertinent actions.

  To avoid cluttering meaningful information with useless audit records and reduce the amount of audit trail administration, only audit the targeted database activities.

- Archive audit records and purge the audit trail.

  After you have collected the required information, archive the audit records of interest and purge the audit trail of this information.

Creating and Deleting the Database Audit Trail Views

This section describes how to create and delete database audit trail views, and includes the following topics:

- Creating the Audit Trail Views
- Deleting the Audit Trail Views

The database audit trail (SYS.AUD$) is a single table in each Oracle database's data dictionary. To help you view meaningful auditing information in this table, several predefined views are provided. They must be created for you to use auditing; you can later delete them if you decide not to use auditing.

Audit trail views are created automatically when you run the script CATALOG.SQL.

Creating the Audit Trail Views

If you decide to use auditing, create the auditing views by connecting as SYS and running the script CATAUDIT.SQL. This script creates the following views:

- STMT_AUDIT_OPTION_MAP
- AUDIT_ACTIONS
- ALL_DEF_AUDIT_OPTS
- DBA_STMT_AUDIT_OPTS
- USER_OBJ_AUDIT_OPTS, DBA_OBJ_AUDIT_OPTS
- USER_AUDIT_TRAIL, DBA_AUDIT_TRAIL

- USER_AUDIT_SESSION, DBA_AUDIT_SESSION
- USER_AUDIT_STATEMENT, DBA_AUDIT_STATEMENT
- USER_AUDIT_OBJECT, DBA_AUDIT_OBJECT
- DBA_AUDIT_EXISTS
- USER_AUDIT_SESSION, DBA_AUDIT_SESSION
- USER_TAB_AUDIT_OPTS

See Also: For information about these views, see the *Oracle8i Reference*.

For examples of audit information interpretations, see "Viewing Database Audit Trail Information" on page 25-17.

Deleting the Audit Trail Views

If you disable auditing and no longer need the audit trail views, delete them by connecting to the database as SYS and running the script file CATNOAUD.SQL. The name and location of the CATNOAUD.SQL script are operating system-dependent.

Managing Audit Trail Information

This section describes various aspects of managing audit trail information, and includes the following topics:

- Events Audited by Default
- Setting Auditing Options
- Enabling and Disabling Database Auditing
- Controlling the Growth and Size of the Audit Trail
- Protecting the Audit Trail

Depending on the events audited and the auditing options set, the audit trail records can contain different types of information. The following information is always included in each audit trail record, provided that the information is meaningful to the particular audit action:

- user name
- session identifier
- terminal identifier

- name of the object accessed
- operation performed or attempted
- completion code of the operation
- date and time stamp

Audit trail records written to the operating system audit trail contain some encodings that are not readable. These can be decoded as follows:

Action Code
This describes the operation performed or attempted. The AUDIT_ACTIONS data dictionary table contains a list of these codes and their descriptions.

Privileges Used
This describes any system privileges used to perform the operation. The SYSTEM_PRIVILEGE_MAP table lists all of these codes, and their descriptions.

Completion Code
This describes the result of the attempted operation. Successful operations return a value of zero, while unsuccessful operations return the Oracle error code describing why the operation was unsuccessful.

Events Audited by Default

Regardless of whether database auditing is enabled,Oracle will always audit certain database-related actions into the operating system audit trail. These events include the following:

instance startup	An audit record is generated that lists the OS user starting the instance, the user's terminal identifier, the date and time stamp, and whether database auditing was enabled or disabled. This is stored in the OS audit trail because the database audit trail is not available until after startup has successfully completed. Recording the state of database auditing at startup also prevents an administrator from restarting a database with database auditing disabled (so they can perform unaudited actions).
instance shutdown	An audit record is generated that lists the OS user shutting down the instance, the user's terminal identifier, the date and time stamp.
connections to the database with administrator privileges	An audit record is generated that lists the OS user connecting to Oracle as SYSOPER or SYSDBA, to provide accountability of users with administrator privileges.

On operating systems that do not make an audit trail accessible to Oracle, these audit trail records are placed in an Oracle audit trail file in the same directory as background process trace files.

Setting Auditing Options

Depending on the auditing options set, audit records can contain different types of information. However, all auditing options generate the following information:

- the user that executed the audited statement
- the action code (a number) that indicates the audited statement executed by the user
- the object or objects referenced in the audited statement
- the date and time that the audited statement was executed

The audit trail does not store information about any data values that might be involved in the audited statement. For example, old and new data values of updated rows are not stored when an UPDATE statement is audited. However, this

specialized type of auditing can be performed on DML statements involving tables by using database triggers.

Oracle allows you to set audit options at three levels:

statement	audits on the type of SQL statement used, such as any SQL statement on a table (which records each CREATE, TRUNCATE, and DROP TABLE statement)
privilege	audits use of a particular system privilege, such as CREATE TABLE
object	audits specific statements on specific objects, such as ALTER TABLE on the EMP table

See Also: For examples of trigger usage for this specialized type of auditing, see "Auditing Through Database Triggers" on page 25-20.

Statement Audit Options

Valid statement audit options that can be included in AUDIT and NOAUDIT statements are listed in the *Oracle8i SQL Reference*.

Shortcuts for Statement Audit Options Shortcuts are provided so that you can specify several related statement options with one word.

Shortcuts are not statement options themselves; rather, they are ways of specifying sets of related statement options with one word in AUDIT and NOAUDIT statements. Shortcuts for system privileges and statement options are detailed in the *Oracle8i SQL Reference.*

Auditing Connections and Disconnections

The SESSION statement option (and CONNECT shortcut) is unique because it does not generate an audit record when a particular type of statement is issued; this option generates a single audit record for each session created by connections to an instance. An audit record is inserted into the audit trail at connect time and updated at disconnect time. Cumulative information about a session such as connection time, disconnection time, logical and physical I/Os processed, and more is stored in a single audit record that corresponds to the session.

See Also: The *Oracle8i SQL Reference* also lists additional audit options not covered by the shortcuts.

Privilege Audit Options

Privilege audit options exactly match the corresponding system privileges. For example, the option to audit use of the DELETE ANY TABLE privilege is DELETE ANY TABLE. To turn this option on, you would use a statement similar to the following example:

```
AUDIT DELETE ANY TABLE
    BY ACCESS
    WHENEVER NOT SUCCESSFUL;
```

Oracle's system privileges are listed in "System Privileges" on page 24-2.

Object Audit Options

The *Oracle8i SQL Reference* lists valid object audit options and the schema object types for which each option is available.

Shortcut for Object Audit Options The ALL shortcut can be used to specify all available object audit options for a schema object. This shortcut is not an option itself; rather, it is a way of specifying all object audit options with one word in AUDIT and NOAUDIT statements.

Enabling Audit Options

The SQL statement AUDIT turns on statement and privilege audit options, and object audit options. To use it to set statement and privilege options, you must have the AUDIT SYSTEM privilege. To use it to set object audit options, you must own the object to be audited or have the AUDIT ANY privilege. Audit statements that set statement and privilege audit options can include a BY clause to specify a list of users or application proxies to limit the scope of the statement and privilege audit options.

You can set any auditing option, and specify the following conditions for auditing:

- WHENEVER SUCCESSFUL / WHENEVER NOT SUCCESSFUL
- BY SESSION / BY ACCESS

A new database session picks up auditing options from the data dictionary when the session is created. These auditing options remain in force for the duration of the database connection. Setting new system or object auditing options causes all

subsequent database sessions to use these options; existing sessions will continue using the audit options in place at session creation.

> **WARNING: The AUDIT statement only specifies auditing options; it does not enable auditing as a whole. To turn auditing on and control whether Oracle generates audit records based on the audit options currently set, set the parameter AUDIT_TRAIL in the database's parameter file.**

See Also: For a complete description of the AUDIT command, see the *Oracle8i SQL Reference*.

For more information about enabling and disabling auditing, see "Enabling and Disabling Database Auditing" on page 25-13.

Enabling Statement Privilege Auditing To audit all successful and unsuccessful connections to and disconnections from the database, regardless of user, BY SESSION (the default and only value for this option), enter the following statement:

```
AUDIT SESSION;
```

You can set this option selectively for individual users also, as in the next example:

```
AUDIT SESSION
BY scott, lori;
```

To audit all successful and unsuccessful uses of the DELETE ANY TABLE system privilege, enter the following statement:

```
AUDIT DELETE ANY TABLE;
```

To audit all unsuccessful SELECT, INSERT, and DELETE statements on all tables and unsuccessful uses of the EXECUTE PROCEDURE system privilege, by all database users, and by individual audited statement, issue the following statement:

```
AUDIT SELECT TABLE, INSERT TABLE, DELETE TABLE,
   EXECUTE PROCEDURE
      BY ACCESS
      WHENEVER NOT SUCCESSFUL;
```

The AUDIT SYSTEM system privilege is required to set any statement or privilege audit option. Normally, the security administrator is the only user granted this system privilege.

Enabling Object Auditing To audit all successful and unsuccessful DELETE statements on the SCOTT.EMP table, BY SESSION (the default value), enter the following statement:

```
AUDIT DELETE ON scott.emp;
```

To audit all successful SELECT, INSERT, and DELETE statements on the DEPT table owned by user JWARD, BY ACCESS, enter the following statement:

```
AUDIT SELECT, INSERT, DELETE
     ON jward.dept
     BY ACCESS
     WHENEVER SUCCESSFUL;
```

To set the default object auditing options to audit all unsuccessful SELECT statements, BY SESSION (the default), enter the following statement:

```
AUDIT SELECT
     ON DEFAULT
     WHENEVER NOT SUCCESSFUL;
```

A user can set any object audit option for the objects contained in the user's schema. The AUDIT ANY system privilege is required to set an object audit option for an object contained in another user's schema or to set the default object auditing options; normally, the security administrator is the only user granted this system privilege.

Disabling Audit Options

The NOAUDIT command turns off the various audit options of Oracle. Use it to reset statement and privilege audit options, and object audit options. A NOAUDIT statement that sets statement and privilege audit options can include the BY USER option to specify a list of users to limit the scope of the statement and privilege audit options.

You can use a NOAUDIT statement to disable an audit option selectively using the WHENEVER clause. If the clause is not specified, the auditing option is disabled entirely, for both successful and unsuccessful cases.

The BY SESSION/BY ACCESS option pair is *not* supported by the NOAUDIT command; audit options, no matter how they were turned on, are turned off by an appropriate NOAUDIT statement.

> **WARNING: The NOAUDIT statement only specifies auditing options; it does not disable auditing as a whole. To turn auditing off and stop Oracle from generating audit records, even though you have audit options currently set, set the parameter AUDIT_TRAIL in the database's parameter file.**

See Also: For a complete syntax listing of the NOAUDIT command, see the *Oracle8i SQL Reference*.

Also see "Enabling and Disabling Database Auditing" on page 25-13.

Disabling Statement and Privilege Auditing

The following statements turn off the corresponding audit options:

```
NOAUDIT session;
NOAUDIT session BY scott, lori;
NOAUDIT DELETE ANY TABLE;
NOAUDIT SELECT TABLE, INSERT TABLE, DELETE TABLE,
    EXECUTE PROCEDURE;
```

The following statements turn off all statement (system) and privilege audit options:

```
NOAUDIT ALL;
NOAUDIT ALL PRIVILEGES;
```

To disable statement or privilege auditing options, you must have the AUDIT SYSTEM system privilege.

Disabling Object Auditing The following statements turn off the corresponding auditing options:

```
NOAUDIT DELETE
   ON emp;
NOAUDIT SELECT, INSERT, DELETE
   ON jward.dept;
```

Furthermore, to turn off all object audit options on the EMP table, enter the following statement:

```
NOAUDIT ALL
```

```
   ON emp;
```

Disabling Default Object Audit Options To turn off all default object audit options, enter the following statement:

```
NOAUDIT ALL
   ON DEFAULT;
```

Note that all schema objects created before this NOAUDIT statement is issued continue to use the default object audit options in effect at the time of their creation, unless overridden by an explicit NOAUDIT statement after their creation.

To disable object audit options for a specific object, you must be the owner of the schema object. To disable the object audit options of an object in another user's schema or to disable default object audit options, you must have the AUDIT ANY system privilege. A user with privileges to disable object audit options of an object can override the options set by any user.

Enabling and Disabling Database Auditing

Any authorized database user can set statement, privilege, and object auditing options at any time, but Oracle does not generate and store audit records in the audit trail unless database auditing is enabled. The security administrator is normally responsible for this operation.

Database auditing is enabled and disabled by the AUDIT_TRAIL initialization parameter in the database's parameter file. The parameter can be set to the following values:

DB	enables database auditing and directs all audit records to the database audit trail
OS	enables database auditing and directs all audit records to the operating system audit trail
NONE	disables auditing (This value is the default.)

After you have edited the parameter file, restart the database instance to enable or disable database auditing as intended.

See Also: For more information about editing parameter files, see the *Oracle8i Reference*.

Controlling the Growth and Size of the Audit Trail

If the audit trail becomes completely full and no more audit records can be inserted, audited statements cannot be successfully executed until the audit trail is purged. Warnings are returned to all users that issue audited statements. Therefore, the security administrator must control the growth and size of the audit trail.

When auditing is enabled and audit records are being generated, the audit trail grows according to two factors:

- the number of audit options turned on
- the frequency of execution of audited statements

To control the growth of the audit trail, you can use the following methods:

- Enable and disable database auditing. If it is enabled, audit records are generated and stored in the audit trail; if it is disabled, audit records are not generated.
- Be very selective about the audit options that are turned on. If more selective auditing is performed, useless or unnecessary audit information is not generated and stored in the audit trail.
- Tightly control the ability to perform object auditing. This can be done two different ways:
  - A security administrator owns all objects and the AUDIT ANY system privilege is never granted to any other user. Alternatively, all schema objects can belong to a schema for which the corresponding user does not have CREATE SESSION privilege.
  - All objects are contained in schemas that do not correspond to real database users (that is, the CREATE SESSION privilege is not granted to the corresponding user) and the security administrator is the only user granted the AUDIT ANY system privilege.

  In both scenarios, object auditing is controlled entirely by the security administrator.

The maximum size of the database audit trail (SYS.AUD$ table) is predetermined during database creation. By default, up to 99 extents, each 10K in size, can be allocated for this table.

You *cannot* move SYS.AUD$ to another tablespace as a means of controlling the growth and size of the audit trail. However, you can modify the default storage parameters (except INITIAL) in SYS.AUD$.

See Also: If you are directing audit records to the operating system audit trail, see your operating system-specific Oracle documentation for more information about managing the operating system audit trail.

For more details on the SYS.AUD$ storage parameters, see the *Oracle8i Reference*.

Purging Audit Records from the Audit Trail

After auditing is enabled for some time, the security administrator may want to delete records from the database audit trail both to free audit trail space and to facilitate audit trail management.

For example, to delete *all* audit records from the audit trail, enter the following statement:

```
DELETE FROM sys.aud$;
```

Alternatively, to delete all audit records from the audit trail generated as a result of auditing the table EMP, enter the following statement:

```
DELETE FROM sys.aud$
     WHERE obj$name='EMP';
```

If audit trail information must be archived for historical purposes, the security administrator can copy the relevant records to a normal database table (for example, using "INSERT INTO table SELECT ... FROM sys.aud$ ...") or export the audit trail table to an operating system file.

Only the user SYS, a user who has the DELETE ANY TABLE privilege, or a user to whom SYS has granted DELETE privilege on SYS.AUD$ can delete records from the database audit trail.

Note: If the audit trail is completely full and connections are being audited (that is, if the SESSION option is set), typical users cannot connect to the database because the associated audit record for the connection cannot be inserted into the audit trail. In this case, the security administrator must connect as SYS (operations by SYS are not audited) and make space available in the audit trail.

See Also: For information about exporting tables, see *Oracle8i Utilities*.

Reducing the Size of the Audit Trail

As with any database table, after records are deleted from the database audit trail, the extents allocated for this table still exist.

If the database audit trail has many extents allocated for it, but many of them are not being used, the space allocated to the database audit trail can be reduced using the following steps:

1. If you want to save information currently in the audit trail, copy it to another database table or export it using the EXPORT utility.
2. Connect as with administrator privileges.
3. Truncate SYS.AUD$ using the TRUNCATE command.
4. Reload archived audit trail records generated from Step 1.

The new version of SYS.AUD$ is allocated only as many extents as are necessary to contain current audit trail records.

> **Note:** SYS.AUD$ is the only SYS object that should ever be directly modified.

Protecting the Audit Trail

When auditing for suspicious database activity, protect the integrity of the audit trail's records to guarantee the accuracy and completeness of the auditing information.

To protect the database audit trail from unauthorized deletions, grant the DELETE ANY TABLE system privilege to security administrators only.

To audit changes made to the database audit trail, use the following statement:

```
AUDIT INSERT, UPDATE, DELETE
    ON sys.aud$
    BY ACCESS;
```

Audit records generated as a result of object audit options set for the SYS.AUD$ table can only be deleted from the audit trail by someone connected with administrator privileges, which itself has protection against unauthorized use. As a final measure of protecting the audit trail, any operation performed while connected with administrator privileges is audited in the operating system audit trail, if available.

See Also: For more information about the availability of an operating system audit trail and possible uses, see your operating system-specific Oracle documentation.

Viewing Database Audit Trail Information

This section offers examples that demonstrate how to examine and interpret the information in the audit trail, and includes the following topics:

- Listing Active Statement Audit Options
- Listing Active Privilege Audit Options
- Listing Active Object Audit Options for Specific Objects
- Listing Default Object Audit Options
- Listing Audit Records
- Listing Audit Records for the AUDIT SESSION Option

You may have to audit a database for the following suspicious activities:

- Passwords, tablespace settings, and quotas for some database users are being altered without authorization.
- A high number of deadlocks are occurring, most likely because of users acquiring exclusive table locks.
- Rows are arbitrarily being deleted from the EMP table in SCOTT's schema.

As an example, say that you suspect the users JWARD and SWILLIAMS of several of these detrimental actions. The database administrator may then issue the following statements (in order):

```
AUDIT ALTER, INDEX, RENAME ON DEFAULT
    BY SESSION;
CREATE VIEW scott.employee AS SELECT * FROM scott.emp;
AUDIT SESSION BY jward, swilliams;
AUDIT ALTER USER;
AUDIT LOCK TABLE
    BY ACCESS
    WHENEVER SUCCESSFUL;
AUDIT DELETE ON scott.emp
    BY ACCESS
    WHENEVER SUCCESSFUL;
```

The following statements are subsequently issued by the user JWARD:

```
ALTER USER tsmith QUOTA 0 ON users;
DROP USER djones;
```

The following statements are subsequently issued by the user SWILLIAMS:

```
LOCK TABLE scott.emp IN EXCLUSIVE MODE;
DELETE FROM scott.emp WHERE mgr = 7698;
ALTER TABLE scott.emp ALLOCATE EXTENT (SIZE 100K);
CREATE INDEX scott.ename_index ON scott.emp (ename);
CREATE PROCEDURE scott.fire_employee (empid NUMBER) AS
  BEGIN
    DELETE FROM scott.emp WHERE empno = empid;
  END;
/

EXECUTE scott.fire_employee(7902);
```

The following sections show the information that can be listed using the audit trail views in the data dictionary.

Listing Active Statement Audit Options

The following query returns all the statement audit options that are set:

```
SELECT * FROM sys.dba_stmt_audit_opts;

USER_NAME            AUDIT_OPTION         SUCCESS         FAILURE
-------------------- -------------------- ----------      ---------
JWARD                SESSION              BY SESSION      BY SESSION
SWILLIAMS            SESSION              BY SESSION      BY SESSION
                     LOCK TABLE           BY ACCESS       NOT SET
```

Notice that the view reveals the statement audit options set, whether they are set for success or failure (or both), and whether they are set for BY SESSION or BY ACCESS.

Listing Active Privilege Audit Options

The following query returns all the privilege audit options that are set:

```
SELECT * FROM sys.dba_priv_audit_opts;

USER_NAME           PRIVILEGE            SUCCESS      FAILURE
------------------- -------------------- ---------    ----------
ALTER USER          BY SESSION           BY SESSION
```

Listing Active Object Audit Options for Specific Objects

The following query returns all audit options set for any objects contained in SCOTT's schema:

```
SELECT * FROM sys.dba_obj_audit_opts
    WHERE owner = 'SCOTT' AND object_name LIKE 'EMP%';

OWNER OBJECT_NAME OBJECT_TY ALT AUD COM DEL GRA IND INS LOC ...
----- ----------- --------- --- --- --- --- --- --- --- --- ...
SCOTT EMP         TABLE      S/S -/- -/- A/- -/- S/S -/- -/- ...
SCOTT EMPLOYEE     VIEW      -/- -/- -/- A/- -/- S/S -/- -/- ...
```

Notice that the view returns information about all the audit options for the specified object. The information in the view is interpreted as follows:

- The character "-" indicates that the audit option is not set.
- The character "S" indicates that the audit option is set, BY SESSION.
- The character "A" indicates that the audit option is set, BY ACCESS.
- Each audit option has two possible settings, WHENEVER SUCCESSFUL and WHENEVER NOT SUCCESSFUL, separated by "/". For example, the DELETE audit option for SCOTT.EMP is set BY ACCESS for successful delete statements and not set at all for unsuccessful delete statements.

Listing Default Object Audit Options

The following query returns all default object audit options:

```
SELECT * FROM all_def_audit_opts;

ALT AUD COM DEL GRA IND INS LOC REN SEL UPD REF EXE
--- --- --- --- --- --- --- --- --- --- --- --- ---
S/S -/- -/- -/- -/- S/S -/- -/- S/S -/- -/- -/- -/-
```

Notice that the view returns information similar to the USER_OBJ_AUDIT_OPTS and DBA_OBJ_AUDIT_OPTS views (see previous example).

Listing Audit Records

The following query lists audit records generated by statement and object audit options:

```
SELECT * FROM sys.dba_audit_object;
```

Listing Audit Records for the AUDIT SESSION Option

The following query lists audit information corresponding to the AUDIT SESSION statement audit option:

```
SELECT username, logoff_time, logoff_lread, logoff_pread,
    logoff_lwrite, logoff_dlock
    FROM sys.dba_audit_session;

USERNAME   LOGOFF_TI LOGOFF_LRE LOGOFF_PRE LOGOFF_LWR LOGOFF_DLO
---------- --------- ---------- ---------- ---------- ----------
JWARD      02-AUG-91       53          2         24    0
SWILLIAMS  02-AUG-91     3337        256        630    0
```

Auditing Through Database Triggers

You can use triggers to supplement the built-in auditing features of Oracle. Although you can write triggers to record information similar to that recorded by the AUDIT command, do so only when you need more detailed audit information. For example, you can use triggers to provide value-based auditing on a per-row basis for tables.

> **Note:** In some fields, the Oracle AUDIT command is considered a *security* audit facility, while triggers can provide a *financial* audit facility.

When deciding whether to create a trigger to audit database activity, consider the advantages that the standard Oracle database auditing features provide compared to auditing by triggers:

- Standard auditing options cover DML and DDL statements regarding all types of schema objects and structures.
- All database audit information is recorded centrally and automatically using the auditing features of Oracle.
- Auditing features enabled using the standard Oracle features are easier to declare and maintain and less prone to errors than are auditing functions defined through triggers.
- Any changes to existing auditing options can also be audited to guard against malicious database activity.

- Using the database auditing features, you can generate records once every time an audited statement is issued (BY ACCESS) or once for every session that issues an audited statement (BY SESSION). Triggers cannot audit by session; an audit record is generated each time a trigger-audited table is referenced.
- Database auditing can audit unsuccessful data access. In comparison, any audit information generated by a trigger is rolled back if the triggering statement is rolled back.
- Connections and disconnections, as well as session activity (such as physical I/Os, logical I/Os, and deadlocks), can be recorded by standard database auditing.

When using triggers to provide sophisticated auditing, normally use AFTER triggers. By using AFTER triggers, you record auditing information after the triggering statement is subjected to any applicable integrity constraints, preventing cases where audit processing is carried out unnecessarily for statements that generate exceptions to integrity constraints.

When you should use AFTER row as opposed to AFTER statement triggers depends on the information being audited. For example, row triggers provide value-based auditing on a per-row basis for tables. Triggers can also allow the user to supply a "reason code" for issuing the audited SQL statement, which can be useful in both row and statement-level auditing situations.

The following trigger audits modifications to the EMP table on a per-row basis. It requires that a "reason code" be stored in a global package variable before the update. The trigger demonstrates the following:

- how triggers can provide value-based auditing
- how to use public package variables

Comments within the code explain the functionality of the trigger.

```
CREATE TRIGGER audit_employee
AFTER INSERT OR DELETE OR UPDATE ON emp
FOR EACH ROW
BEGIN
/* AUDITPACKAGE is a package with a public package
   variable REASON. REASON could be set by the
   application by a command such as EXECUTE
   AUDITPACKAGE.SET_REASON(reason_string). Note that a
   package variable has state for the duration of a
   session and that each session has a separate copy of
   all package variables. */
IF auditpackage.reason IS NULL THEN
   raise_application_error(-20201,'Must specify reason with ',
```

```
   'AUDITPACKAGE.SET_REASON(reason_string)');
END IF;

/* If the above conditional evaluates to TRUE, the
    user-specified error number and message is raised,
    the trigger stops execution, and the effects of the
    triggering statement are rolled back. Otherwise, a
    new row is inserted into the pre-defined auditing
    table named AUDIT_EMPLOYEE containing the existing
    and new values of the EMP table and the reason code
    defined by the REASON variable of AUDITPACKAGE. Note
    that the "old" values are NULL if triggering
    statement is an INSERT and the "new" values are NULL
    if the triggering statement is a DELETE. */
INSERT INTO audit_employee VALUES
    (:old.ssn, :old.name, :old.job_classification, :old.sal,
    :new.ssn, :new.name, :new.job_classification, :new.sal,
    auditpackage.reason, user, sysdate );
END;
```

Optionally, you can also set the reason code back to NULL if you want to force the reason code to be set for every update. The following AFTER statement trigger sets the reason code back to NULL after the triggering statement is executed:

```
CREATE TRIGGER audit_employee_reset
   AFTER INSERT OR DELETE OR UPDATE ON emp
BEGIN
    auditpackage.set_reason(NULL);
END;
```

The previous two triggers are both fired by the same type of SQL statement. However, the AFTER row trigger is fired once for each row of the table affected by the triggering statement, while the AFTER statement trigger is fired only once after the triggering statement execution is completed.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W