

AT&T 585-350-909
Issue 1
Comcode 407298603

ORACLE⁷TM SERVER ADMINISTRATOR'S GUIDE

Part Number 6694-70-1292
December 1992

ORACLE[®]

Cooperative Server Technology for Transparent Data Sharing

ORACLE7 Server Administrator's Guide

Part Number 6694-701292

December 1992

Contributing Authors: Steven Bobrowski, with Eric Armstrong, Cynthia Closkey, Brian Linden

Contributors: Richard Allen, Dave Anderson, Lori Asher, Andre Bakker, Bill Bridge, Atif Chaudry, Gray Clossman, Jeff Cohen, Dave Ensor, Stephan Faris, John Frazzini, Gloria Gatlin, Anurag Gupta, Gary Hallmark, Chris Harmon, Michael Hartstein, Terry Hayes, Brian Hirano, Ken Jacobs, Sandeep Jain, Robert Jenkins, Jr., Kevin Jernigan, Derry Kabcenell, Michael Kennedy, Jonathan Klein, R. Kleinro, Robert Kooi, Andy Laursen, Lefty Leverenz, Walter Lindsay, Juan Loaiza, Brom Mahbod, William Maimone, Andrew Mendelsohn, Mark Moore, Edward Peeler, Mark Porter, Maria Pratt, Darryl Presley, Tuomas Pystynen, Brian Quigley, Patrick Ritto, Mark Sinykin, Gordon Smith, Leng Leng Tan, Mustafa Vahanvaty, Joyo Wijaya, Beth Winslow, Kevin Wharton

Copyright © 1992 Oracle Corporation. All rights reserved.

This software contains proprietary information of Oracle Corporation; it is provided under a license agreement containing restrictions on use and disclosure and is also protected by copyright law. Reverse engineering of the software is prohibited.

If this software / documentation is delivered to a U.S. Government Agency of the Department of Defense, then it is delivered with Restricted Rights and the following legend is applicable:

RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of DFARS 252.227-7013, Rights in Technical Data and Computer Software (October 1988).

Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

If this software / documentation is delivered to a U.S. Government Agency not within the Department of Defense, then it is delivered with "Restricted Rights," as defined in FAR 52.227-14, Rights in Data - General, including Alterate III (June 1987).

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. Oracle Corporation does not warrant that this document is error-free.

ORACLE, Oracle*Mail, SQL*DBA, SQL*Forms, SQL*Loader, SQL*Net, SQL*Plus, and SQL*ReportWriter are registered trademarks of Oracle Corporation.

ORACLE7, Oracle*Mail, ORACLE Parallel Server, and PL/SQL are trademarks of Oracle Corporation.

All trade names referenced are the service mark, trademark, or registered trademark of the respective manufacturer.



PREFACE

This Guide describes how to manage the ORACLE7 Server, a relational database management system (RDBMS).

Information in this Guide applies to the ORACLE7 Server, running on all operating systems. It provides information about the base ORACLE Server product and its special options, including the following:

- procedural option
- distributed option
- Parallel Server option

Any chapter of this Guide that applies only to a particular option is clearly indicated on the first page of the chapter.

Audience

This Guide is written for those responsible for administering the operation of an ORACLE database system. These persons, 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.

Knowledge Assumed of the Reader

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.

As a prerequisite, **all readers should read the first chapter of the *ORACLE7 Server Concepts Manual***, “A Technical Introduction to the ORACLE Server.” This chapter is a comprehensive introduction to the concepts and terminology used throughout this Guide.

Readers Interested in Installation and Migration Information

Administrators are frequently involved in installing the ORACLE Server software and migrating existing ORACLE databases to newer formats (for example, Version 6 databases to ORACLE7 format). This Guide is not an installation or migration manual. If your primary interest is installation, refer to the installation or user’s guide for your particular operating system. If your primary interest is database or application migration, refer to the *ORACLE7 Server Migration Guide*.

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 refer to the *ORACLE7 Server Application Developer’s Guide* and to the documentation for the tool or language product they are using to develop ORACLE database applications.

How This Guide Is Organized

This Guide is divided into parts and chapters, as described below.

Part I Basic Database Administration

Chapter 1: Understanding Database Administration

This chapter outlines the responsibilities and concerns of the database administrator and describes the default ORACLE administrator accounts that can be used to administer a database.

Chapter 2: Creating a Database

This chapter describes the steps for creating a database in preparation for its use. (This information is intended to supplement, not substitute for, information in the installation or user's guides.)

Chapter 3: Starting Up and Shutting Down

This chapter describes how to control the accessibility of an ORACLE database system. It also describes how to initialize parameter settings.

Part II ORACLE Server Configuration

Chapter 4: Managing ORACLE Processes

This chapter describes how to manage the processes that make up an ORACLE instance, including how to set up the different process configurations available for ORACLE.

Chapter 5: Managing the Online Redo Log

This chapter explains how to maintain the online redo log, including managing mirrored redo log groups and their members.

Chapter 6: Managing Control Files

This chapter explains how to create, mirror, replace, and drop control files for a database.

Part III Database Storage

Chapter 7: Managing the Size and Files of the Database

This chapter discusses how to manage the physical storage space in an ORACLE database by managing tablespaces and data files. This includes adding and dropping tablespaces and data files, and taking tablespaces and data files on and offline.

Chapter 8: Managing Schema Objects

This chapter discusses how to manage the objects that contain data (tables, views, numeric sequences, and synonyms, as well as indexes and clusters). This includes controlling how data is stored and how storage space is allocated for and consumed by objects within an ORACLE database. It explains how to create, alter, and drop these objects, and how to perform other standard administrative functions on them.

Chapter 9: Managing Rollback Segments

This chapter discusses how to create, alter, manage, and drop rollback segments in the database, and offers guidelines for their management.

Part IV Database Security

Chapter 10: Establishing Security Policies

This chapter discusses how to create a security policy that uses the security features of the ORACLE Server.

Chapter 11: Managing Users and Resources

This chapter describes how to manage database users and limit the use of system resources.

Chapter 12: Controlling User Privileges and Roles

This chapter discusses how to administer system and object security.

Chapter 13: Auditing Database Use

This chapter discusses how to monitor database activity using the auditing feature of the ORACLE Server.

Part V Distributed Processing and Distributed Databases

Chapter 14: Administering Client-Server Systems

This chapter discusses setting up distributed processing environments that include the ORACLE Server.

Chapter 15: Managing Distributed Databases

This chapter explains how to administer a distributed database system, including setting up global names, using database links, and resolving in-doubt distributed transactions.

Chapter 16: Managing Table Snapshots

This chapter discusses how to replicate data among nodes of a distributed database, and how to refresh such snapshots.

Part VI Database Backup and Recovery

Chapter 17: Archiving Redo Information

This chapter explains how to archive the redo log files for a database, and how to manage the archived files.

Chapter 18: Backing Up a Database

This chapter discusses how to protect an ORACLE database from possible failures.

Chapter 19: Recovering a Database

This chapter explains how to recover a database from failure using backups of the database.

Part VII Database and Instance Tuning

Chapter 20: Understanding the Tuning Process

This chapter outlines the steps of the database tuning process. It tells you how to follow the steps described in subsequent chapters, and introduces issues discussed in each step.

Chapter 21: Tuning Memory Allocation

This chapter helps you allocate memory to ORACLE data structures most efficiently. These structures include SQL and PL/SQL areas and the buffer cache.

Chapter 22: Tuning I/O

This chapter discusses how to avoid I/O bottlenecks that can reduce performance. In this chapter, you learn to reduce disk contention, allocate space in data blocks, and avoid having ORACLE manage space dynamically.

Chapter 23: Tuning Contention

This chapter discusses the problems caused by contention and helps you recognize when these problems occur. In this chapter, you learn to reduce contention for rollback segments and redo log buffer latches.

Chapter 24: Additional Tuning Considerations

This chapter discusses specific performance-related aspects of the ORACLE Server, including sorts, free lists, and checkpoints.

Reference

Appendix A: Initialization Parameter Files

This appendix is a reference to the parameter files and individual parameters that are used to initialize various settings when an ORACLE database system is started.

Appendix B: Data Dictionary Reference

This appendix is a reference to the many data dictionary views of an ORACLE database.

Appendix C: National Language Support

This appendix explains how to configure ORACLE to handle data and interface with users using various languages.

Appendix D: Database Limits

This appendix is a quick reference to many limits of values associated with various database functions and objects.

Appendix E: Operating System Specific Information

This appendix is a summary of all the operating system specific references contained within this Guide.

How to Use This Guide

Every reader of this Guide **must** read Chapter 1 of the *ORACLE7 Server Concepts Manual*, “A Technical Introduction to the ORACLE Server.” This overview of the concepts and terminology related to ORACLE provides a foundation for the more detailed information in this Guide. The rest of the *ORACLE7 Server Concepts Manual* explains the ORACLE architecture and features and how they operate in more detail.

Conventions Used in This Guide

The following sections explain the conventions used in this Guide.

Text of the Guide

The following section explains the conventions used within the text of this Guide:

UPPERCASE WORDS

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 of the rollback segment must be included in the `ROLLBACK_SEGMENTS` parameter of the parameter file

Italicized Words

Italicized words within text are used to indicate the first occurrence and definition of a term, as in the following example:

A *database* is a collection of data to be treated as a unit. The general purpose of a database is to store and retrieve related information, as needed.

Italicized words are also used to indicate book titles and to highlight names of performance statistics.

Bold Words

Bolding within text is used to call special attention to important information. For example:

In summary, remember that this procedure provides a reasonable **estimate** of a table’s size, not an exact number of blocks or bytes.

Examples of the SQL*DBA Interface

Throughout this Guide, examples of the dialog boxes and menus of SQL*DBA, your primary utility for managing an ORACLE database, are provided. Illustrations of the character mode SQL*DBA screen are provided; however, the actual appearance of your screen may differ, depending on your system's user interface. (SQL*DBA is documented fully in the *ORACLE7 Server Utilities User's Guide*.)

Examples of Commands and Statements

SQL, SQL*DBA, and SQL*Plus commands and statements appear separated from the text of paragraphs in a fixed-width font:

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

Punctuation , ' "	Example statements may include punctuation such as commas or quotation marks. All punctuation given in example statements is required. All example statements are terminated with a semicolon. Depending on the application being used, a semi-colon or other terminator may or may not be required to end a statement.
Uppercase Words INSERT, SIZE	Uppercase words in example statements are used to indicate the keywords within ORACLE SQL. However, note that when issuing statements, keywords are not case-sensitive.
Lowercase Words emp, users2.ora	Lowercase words in example statements are used to indicate words supplied only for the context of the example. For example, lowercase words may indicate the name of a table, column, or file. Some operating systems are case sensitive, so refer to your installation or user's guide to determine whether you must pay attention to case.

Special Icons

Two special icons are provided to alert you to particular information within the body of this Guide and other manuals:



- The lightbulb highlights suggestions and practical tips that could save time, make procedures easier, and so on.
- The warning symbol highlights text that warns you of actions that could be particularly damaging or fatal to your system.

Related Publications



Admin

The *ORACLE7 Server Administrator's Guide* is one of many books used to explain various parts of an ORACLE database system.

In many sections of this Guide, the information is general because the Guide is written to apply to ORACLE working under any operating system and configuration. Therefore, you will also find many references in this book to important information in the related publications. To clearly indicate an area in this Guide that maybe supported by information in a related book, a corresponding icon is placed in the margin of the page, adjacent to the text referring to the supporting book. The related books (and corresponding icons) referred to in this Guide are included below.



Concepts

- For general information about the ORACLE7 Server and how it works, see the *ORACLE7 Server Concepts Manual*, Part No. 6693-70.



AppDev

- For information about developing database applications within the ORACLE7 Server, see the *ORACLE7 Server Application Developer's Manual*, Part No. 6695-70.



Migrate

- For the procedures for migrating a previous version of ORACLE to ORACLE7, see the *ORACLE7 Server Migration Guide*, Part No. 6617-70.



IUG

- For information specific to the ORACLE Server working on your host operating system, refer to your installation or user's guide (specific book titles vary by operating system) and system release bulletins, if available.



SQL

- For reference information on ORACLE's SQL commands and functions, refer to the *ORACLE7 Server SQL Language Reference Manual*, Part No. 778-70.



PL/SQL

- For information about ORACLE'S procedural language extension to SQL, PL/SQL, refer to the *PL/SQL User's Guide and Reference*, Part No. 800-20.



OCI

- For information about programming applications that use the ORACLE Call Interfaces, see the *ORACLE Call Interfaces Programmer's Guide*, Part No. 5411-70.



Utilities

- For information about the utilities bundled with the ORACLE Server, including SQL*DBA, Export, Import, and SQL*Loader, refer to the *ORACLE7 Server Utilities User's Guide*, Part No. 3602-70.



Parallel



Trusted



Message



SQL*Net

- For information specific to administering the ORACLE Parallel Server, refer to the *ORACLE7 Parallel Server Administrator's Guide*, Part No. 5990-70.
- For information specific to administering ORACLE's multilevel secure database management system product, Trusted ORACLE, refer to the *Trusted ORACLE7 Server Administrator's Guide*, Part No. 6610-70.
- For more information about ORACLE messages and codes, refer to the *ORACLE7 Server Messages and Codes Manual*, Part No. 3605-70.
- For information about SQL*Net, see the SQL*Net documentation, which includes the *SQL*Net V2 Administrator's Guide*, Part No. 6545-20; the *MultiProtocol Interchange Administrator's Guide*, Part No. 6544-10; and the *Oracle Network Products Messages Manual*, Part No. 6543-00.

Oracle Corporation also publishes a file commonly named README.DOC. This file is available on your distribution media and describes differences between minor releases of Oracle software that are not accompanied by new manuals. The exact name and location of this file might vary, depending on your operating system. Read this file to learn about changes to the software that are not described in the regular manuals.

Your Comments Are Welcome

We value and appreciate your comments as an ORACLE user and reader of the manuals. As we write, revise, and evaluate, your opinions are the most important input we receive. At the back of this Guide is a Reader's Comment Form, which we encourage you to use to tell us what you like and dislike about this Guide or other ORACLE manuals. If the form has been used or you would like to contact us, please use the following address or FAX number, or call us at (415) 506-7000.

ORACLE7 Server Documentation Manager
Oracle Corporation
500 Oracle Parkway
Redwood Shores, CA 94065
FAX: (415) 506-7200

CONTENTS

PART I

BASIC DATABASE ADMINISTRATION

Chapter 1

Administering ORACLE Databases	1 - 1
Database Administrator Responsibilities	1 - 2
Security Officers	1 - 2
Responsibilities of Other Types of Users	1 - 3
Application Developers	1 - 3
Database Users	1 - 3
Database Administrator Security and Privileges	1 - 3
The Database Administrator's Operating System Account	1 - 3
Connecting as INTERNAL	1 - 4
Using a Password for INTERNAL	1 - 4
INTERNAL and Non-Secure Connections	1 - 4
Alternatives to INTERNAL: OSOPER and OSDBA	1 - 5
The DBA Role	1 - 5
Database Administrator Usernames	1 - 6
SW	1 - 6
SYSTEM	1 - 6
Database Administrator Utilities	1 - 7
SQL*DBA	1 - 7
SQL*Loader	1 - 7
Export and Import Utilities	1 - 7

Initial Priorities of a Database Administrator	1 - 8
Install the ORACLE Software	1 - 8
Evaluate the Database Server Hardware	1 - 8
Plan the Database	1 - 9
Create and Open the Database	1-10
Implement the Database Design	1-10
Back up the Database	1-10
Enroll System Users	1-10
Tune Database Performance	1-11
Identifying ORACLE Software Releases	1-11
Version Number	1-11
Maintenance Release Number	1-11
Patch Release Number	1-11
Port-Specific Patch Release Number	1-11
Examples of Release Numbers	1-12
Versions of Other Oracle Software	1-12
Checking Your Current Release Number	1-12
Chapter 2	
Creating a Database	2 - 1
Preparing to Create a Database	2 - 2
Creation Prerequisites	2 - 3
Using an Initial Database	2 - 3
Migrating an Older Version of the Database	2 - 3
Creating an ORACLE Database	2 - 4
Step 1: Back Up Existing Databases	2 - 4
Step 2: Create Parameter Files	2 - 4
Step 3: Edit New Parameter Files	2 - 5
DB_NAME and DB_DOMAIN	2 - 5
CONTROL_FILES	2 - 5
DB_BLOCK_SIZE	2 - 6
DB_BLOCK_BUFFERS	2 - 7
PROCESSES	2 - 7

ROLLBACK_SEGMENTS	2 - 7
LICENSE_MAX_SESSIONS, LICENSE_SESSION_ - WARNING, and LICENSE_MAX_USERS	2 - 8
Step 4: Check the Instance Identifier	2 - 9
Step 5: Start SQL*DBA and Connect as INTERNAL	2 - 9
Step 6: Start an Instance	2 - 9
Step 7: Create the Database	2 - 9
Step 8: Back Up the Database	2-11
Troubleshooting Database Creation	2-11
After Database Creation	2-11
Initial Tuning Guidelines	2-12
Allocating Rollback Segments	2-12
Choosing Sizes for Rollback Segments	2-12
Distributing I/O	2-12
Creating the Data Dictionary	2-13
Creating Additional Data Dictionary Structures	2-17
Chapter 3	
Starting Up and Shutting Down	3 - 1
General Startup Procedures	3 - 2
Preparing to Start an Instance	3 - 2
Starting SQL*DBA and Connecting as INTERNAL	3 - 2
Specifying a Database Name	3 - 2
Specifying a Parameter Filename	3 - 3
Starting an Instance Without Mounting a Database	3 - 3
Starting an Instance and Mounting a Database	3 - 3
Starting an Instance, and Mounting and Opening a Database	3 - 4
Restricting Access to a Database at Startup	3 - 4
Forcing an Instance to Start	3 - 4
Starting an Instance, Mounting a Database, and Starting Complete Media Recovery	3 - 5
Starting in Exclusive or Parallel Mode	3 - 5
Miscellaneous Startup Topics	3 - 5
Automatic Database Startup at Operating System Start	3 - 5

Starting Remote Instances	3 - 6
Examples of Database and Instance Startup	3 - 6
The Startup Instance Dialog	3 - 6
The STARTUP Command	3 - 6
Troubleshooting Startup	3 - 7
Unavailable Control Files	3 - 7
Unavailable Database or Redo Log Files	3 - 7
Altering Database Availability	3 - 7
Mounting a Database to an Instance	3 - 7
The Mount Database Menu	3 - 8
Opening a Closed Database	3 - 9
Restricting Access to an Open Database	3 - 9
Shutting Down a Database	3-10
Connecting as INTERNAL	3-10
Shutting Down a Database Under Normal Conditions	3-10
Shutting Down a Database Immediately	3-11
Aborting an Instance	3-11
Examples of Database Shutdown	3-12
The Shut Down Menu	3 - 12
The SHUTDOWN Command	3-13
Using Parameter Files	3-13
The Sample Parameter File	3-13
Parameter File Character Set	3 - 14
The Number of Parameter Files	3 - 14
The Location of the Parameter File in Distributed Processing Environments	3 - 14
Editing Parameter Files	3 - 14

Chapter 4

Managing ORACLE Processes	4-1
Starting Single-Process and Multiple-Process Instances	4-2
Setting the Process Mode	4-2
Starting ORACLE with Only Dedicated Servers	4-2
Starting ORACLE with Multi-Threaded Servers	4-3
Allocating Additional Space in the Shared Pool for Shared Server	4-3
Setting the Listener Process Address (MTS_LISTENER_ADDRESS)	4-3
Specifying Service Names for Dispatchers (MTS_SERVICE)	4-4
Setting the Initial Number of Dispatchers	4-4
Setting the Maximum Number of Dispatchers	4-5
Setting the Initial Number of Shared Server Processes	4-6
Setting the Maximum Number of Shared Server Processes	4-6
Changing the Minimum Number of Shared Server Processes	4-6
Privileges Required to Change the Minimum Number of Shared Servers	4-7
Adding and Removing Dispatcher Processes	4-7
Privileges Required to Change the Number of Dispatcher Processes	4-9
Deciding When to Connect to a Dedicated Server Process	4-9
Managing ORACLE Processes	4-9
Monitoring the Processes of an ORACLE Instance	4-9
Viewing and Monitoring Locks	4-9
Using Dynamic Performance Tables	4-10
Identifying ORACLE Background Processes from the Operating System	4-10
Trace Files, the ALERT File, and Background Processes	4-11
Using the Trace Files	4-12
Specifying the Location of Trace Files	4-12
Controlling the Size of Trace Files	4-12

	Controlling When ORACLE Writes to Trace Files	4 - 12
	Starting the Checkpoint Process	4 - 13
	Killing sessions	4 - 13
Chapter 5	Managing the Online Redo Log	5 - 1
	Establishing Policies for the Online Redo Log	5 - 2
	Mirror the Online Redo Log	5 - 2
	Appropriately Place Online Redo Log Members	5 - 2
	Appropriately Set the Size of Online Redo Log Members	5 - 2
	Choose the Number of Online Log Files	5 - 3
	Creating Online Redo Log Groups	5 - 4
	Privileges Required to Create Online Redo Log Groups	5 - 5
	Creating Online Redo Log Members	5 - 6
	Renaming and Relocating Online Redo Log Members	5 - 7
	Privileges Required to Rename or Relocate Online Redo Log Members	5 - 9
	Dropping Online Redo Log Groups	5 - 9
	Privileges Required to Drop Online Redo Log Groups	5 - 10
	Dropping Online Redo Log Members	5 - 11
	Privileges Required to Drop Online Redo Log Members	5 - 12
	Controlling Checkpoints and Log Switches	5 - 13
	Setting Database Checkpoint Intervals	5 - 13
	Setting LOG_CHECKPOINT_INTERVAL	5 - 13
	Setting LOG_CHECKPOINT_TIMEOUT	5 - 14
	Forcing a Log Switch	5 - 14
	Privileges Required to Force a Log Switch	5 - 15
	Forcing a Fast Database Checkpoint Without a Log Switch	5 - 15
	Privileges Required to Force a Checkpoint	5 - 15
	Listing Information about the Online Redo Log	5 - 16

Chapter 6	Managing Control Files	6 - 1
	Guidelines for Control Files	6 - 2
	Name Control Files	6 - 2
	Mirror Control Files on Different Disks	6 - 2
	Place Control Files Appropriately	6 - 3
	Manage Control File Size	6 - 3
	Creating Control Files	6 - 3
	Creating Initial Control Files	6 - 3
	Creating Additional Copies of the Control File, and Renaming and Relocating Control Files	6 - 4
	Creating a New Control File	6 - 4
	Checking for Missing or Extra Files	6 - 6
	Handling Errors During CREATE CONTROLFILE	6 - 6
	Dropping Control Files	6 - 7
 PART III	 DATABASE STORAGE	
Chapter 7	Managing the Size and Files of the Database	7 - 1
	Policies for Managing Tablespaces and Data Files	7 - 2
	Use Multiple Tablespaces	7 - 2
	Number of Data Files	7 - 2
	Set the Size of Data Files	7 - 3
	Place Data Files for Maximum Performance	7 - 4
	Store Data Files Separately From Redo Log Files	7 - 4
	Specify Tablespace Storage Parameters	7 - 4
	Assign Tablespace Quotas to Users	7 - 5
	Initially Creating Tablespaces and Data Files	7 - 5
	Creating Additional Tablespaces	7 - 5
	Privileges Required to Create Tablespaces	7 - 7
	Adding Data Files to a Tablespace	7 - 7
	Privileges Required to Add Data Files	7 - 8
	Altering Storage Settings for Tablespaces	7 - 8
	Privileges Required to Alter Tablespace Storage Settings	7 - 9

Altering Tablespace Availability	7 - 9
Bringing Tablespaces Online	7 - 9
Privileges Required to Bring Tablespaces Online	7 - 10
Taking Tablespaces Offline	7 - 10
Privileges Required to Take Tablespaces Offline	7 - 11
Bringing Data Files Online and Offline	7 - 12
Bringing Data Files Online or Offline in ARCHIVELOG Mode	7 - 12
Taking Data Files Offline in NOARCHIVELOG Mode	7 - 12
Privileges Required to Take a Data File On- or Offline	7 - 13
Renaming and Relocating Data Files	7 - 13
Renaming and Relocating Data Files for a Single Tablespace	7 - 13
Privileges Required to Rename Data Files of a Single Tablespace	7 - 14
Renaming and Relocating Data Files for Multiple Tablespaces	7 - 14
Privileges Required to Rename Data Files of Multiple Tablespaces	7 - 15
Example of Relocating a Data File	7 - 15
Dropping Tablespaces and Data Files	7 - 17
Privileges Required to Drop Tablespaces	7 - 18
Viewing Information About Tablespaces and Data Files	7 - 19
Chapter 8	
Managing Schema Objects	8 - 1
Guidelines for Managing Schema Objects	8 - 2
Managing the Space Usage of Data Blocks	8 - 2
Specifying PCTFREE	8 - 2
Specifying PCTUSED	8 - 3
Choosing Associated PCTUSED and PCTFREE Values	8 - 4
Examples of Choosing PCTFREE and PCTUSED Values	8 - 4
Setting Storage Parameters	8 - 5
Setting INITRANS and MAXTRANS	8 - 8
Setting Default Storage Parameters for Segments in a Tablespace	8 - 9
Setting Storage Parameters for Data Segments	8 - 9

Setting Storage Parameters for Index Segments	8 - 9
Changing Values for Storage Parameters	8 - 10
Understanding Precedence in Storage Parameters	8 - 10
Storage Parameter Example	8 - 10
Understanding Space Use of Datatypes	8 - 11
Character Datatypes	8 - 11
NUMBER Datatype	8 - 12
DATE Datatype	8 - 12
LONG Datatype	8 - 12
RAW and LONG RAW Datatypes	8 - 12
ROWIDs and the ROWID Datatype	8 - 12
MLSLABEL Datatype	8 - 13
Summary of ORACLE Datatypes	8 - 14
Managing Tables	8 - 15
Guidelines for Managing Tables	8 - 15
Design Tables Before Creation	8 - 15
Specify How Data Block Space Is to Be Used	8 - 15
Specify Transaction Entry Parameters	8 - 15
Specify the Location of Each Table	8 - 15
Estimate Table Size and Set Storage Parameters	8 - 16
Plan for Very Large Tables	8 - 17
Calculating Space Required by Non-Clustered Tables	8 - 17
Step 1: Calculate Total Block Header Size	8 - 18
Step 2: Calculate Available Data Space Per Data Block	8 - 19
Step 3: Calculate Combined Data Space	8 - 19
Step 4: Calculate Total Average Row Size	8 - 21
Step 5: Calculate Average Rows Per Block	8 - 22
Step 6: Calculate Number of Blocks and Bytes	8 - 22
Space Requirements for Tables in Use	8 - 22
Creating Tables	8 - 22
Privileges Required to Create a Table	8 - 23
Altering Tables	8 - 23

Privileges Required to Alter a Table	8 - 2 5
Manually Allocating Storage for a Table	8 - 2 5
Dropping Tables	8 - 2 5
Privileges Required to Drop a Table	8 - 2 5
Managing Views	8 - 2 6
Creating Views	8 - 2 6
Expansion of Defining Queries at View Creation Time	8 - 2 7
Creating Views with Errors	8 - 2 7
Privileges Required to Create a View	8 - 2 7
Replacing Views	8 - 2 8
Privileges Required to Replace a View	8 - 2 9
Dropping Views	8 - 2 9
Privileges Required to Drop a View	8 - 2 9
Managing Sequences	8 - 2 9
Creating Sequences	8 - 2 9
Privileges Required to Create a Sequence	8 - 3 0
Altering Sequences	8 - 3 0
Privileges Required to Alter a Sequence	8 - 3 0
Initialization Parameters Affecting Sequences	8 - 3 0
Dropping Sequences	8 - 3 1
Privileges Required to Drop a Sequence	8 - 3 1
Managing Synonyms	8 - 3 1
Creating Synonyms	8 - 3 1
Privileges Required to Create a Synonym	8 - 3 1
Dropping Synonyms	8 - 3 1
Privileges Required to Drop a Synonym	8 - 3 2
Managing Indexes	8 - 3 2
Guidelines for Managing Indexes	8 - 3 2
Create Indexes After Inserting Table Data	8 - 3 2
Index the Correct Tables and Columns	8 - 3 3
Limit the Number of Indexes per Table	8 - 3 3
Specify Transaction Entry Parameters	8 - 3 3

Specify Data Block Space Use	8-33
Specify the Tablespace for Each Index	8-34
Estimate Index Size and Set Storage Parameters	8-34
Consider Indexes Before Disabling or Dropping Constraints	8-35
Calculating Space for Indexes	8-35
Step 1: Calculate the Total Block Header Size	8-35
Step 2: Calculate Available Data Space Per Data Block	8-36
Step 3: Calculate Combined Column Lengths of an Index Value	8-36
Step 4: Calculate Total Average Index Value Size	8-36
Step 5: Calculate Number of Blocks and Bytes	8-37
Temporary Space Required for Index Creation	8-38
Creating Indexes	8-38
Creating an Index Associated with a Constraint	8-38
Creating an Index Explicitly	8-39
Privileges Required to Create an Index	8-39
Altering an Index	8-39
Privileges Required to Alter an Index	8-40
Monitoring Space Use of Indexes	8-40
Dropping Indexes	8-40
Dropping an Index Associated with an Integrity Constraint	8-41
Dropping an Index Created Explicitly	8-41
Privileges Required to Drop An Index	8-41
Managing Clusters, Clustered Tables, and Cluster Indexes	8-41
Guidelines for Managing Clusters	8-41
Choose Appropriate Tables to Cluster	8-41
Choose Appropriate Columns for the Cluster Key	8-42
Specify How Data Block Space Is to Be Used	8-42
Specify the Space Required by an Average Cluster Key and Its Associated Rows	8-42
Specify the Location of Each Cluster and Cluster Index	8-43

Estimate Cluster Size and Set Storage Parameters	8-43
Calculating Space Required by Clusters	8-43
Step 1: Calculate Total Block Header Size	8-44
Step 2: Calculate Available Data Space Per Data Block	8-45
Step 3: Calculate Space Required by a Row	8-45
Step 4: Calculate Total Average Row Size	8-46
Step 5: Calculate Average Cluster Block Size	8-47
Step 6: Calculate Total Number of Blocks	8-48
Space Requirements for Clustered Tables in Use	8-49
Creating Clusters, Clustered Tables, and Cluster Indexes	8-49
Privileges Required to Create a Cluster, Clustered Table, and Cluster Index	8-51
Altering Clusters	8-51
Privileges Required to Alter a Cluster	8-52
Altering Clustered Tables and Cluster Indexes	8-52
Manually Allocating Storage for a Cluster	8-52
Dropping Clusters, Clustered Tables, and Cluster Indexes	8-52
Privileges Required to Drop a Cluster	8-54
Managing Hash Clusters and Clustered Tables	8-54
Deciding When to Use Hashing	8-54
Calculating Space Required by Hash Clusters	8-55
Creating Hash Clusters and Clustered Tables	8-56
Controlling Space Use Within a Hash Cluster	8-57
Altering Hash Clusters	8-60
Dropping Hash Clusters	8-60
Miscellaneous Management for Schema Objects	8-60
Creating Multiple Tables and Views in One Operation	8-60
Privileges Required to Create Multiple Schema Objects	8-61
Renaming Schema Objects	8-61
Privileges Required to Rename an Object	8-62
Analyzing Tables, Indexes, and Clusters	8-62

Updating and Deleting Statistics for Tables, Indexes, and Clusters	8 - 6 2
Validating Tables, Indexes, and Clusters	8 - 6 6
Listing Chained Rows of Tables and Clusters	8 - 6 6
Privileges Required to Analyze a Table, Index, or Cluster	8 - 6 7
Truncating Tables and Clusters	8 - 6 7
Privileges and Prerequisites Required to Truncate a Table or Cluster	8 - 6 8
Enabling and Disabling Triggers	8 - 6 9
Disabling Triggers	8 - 6 9
Enabling Triggers	8 - 6 9
Privileges Required to Enable and Disable Triggers	8 - 7 0
Managing Integrity Constraints	8 - 7 0
Understanding Indexes Associated with Constraints	8 - 7 1
Enabling and Disabling Integrity Constraints Upon Definition	8 - 7 1
Enabling and Disabling Existing Integrity Constraints	8 - 7 2
Dropping Integrity Constraints	8 - 7 3
Reporting Constraint Exceptions	8 - 7 3
Managing Object Dependencies	8 - 7 5
Manually Recompiling Views	8 - 7 7
Manually Recompiling Procedures and Functions	8 - 7 7
Manually Recompiling Packages	8 - 7 7
Manually Recompiling Triggers	8 - 7 7
Managing Object Name Resolution	8 - 7 8
Changing Storage Parameters for the Data Dictionary	8 - 7 9
Structures in the Data Dictionary	8 - 7 9
Errors that Require Changing Data Dictionary Storage	8 - 8 1
Displaying Information About Schema Objects	8 - 8 1

Chapter 9	Managing Rollback Segments	9 - 1
	Guidelines for Managing Rollback Segments	9 - 2
	Use Multiple Rollback Segments	9 - 2
	Add a Rollback Segment to the SYSTEM Tablespace	9 - 2
	Choose Between Public and Private Rollback Segments	9 - 3
	Specify Rollback Segments to Acquire Automatically	9 - 3
	Set Rollback Segment Sizes Appropriately	9 - 3
	Create Rollback Segments with Many Extents of Equal Size	9 - 5
	Set an Optimal Number of Extents for Each Rollback Segment	9 - 5
	Set the Storage Location for Rollback Segments	9 - 6
	Creating Rollback Segments	9 - 7
	Privileges Required to Create Rollback Segments	9 - 8
	Bringing New Rollback Segments Online	9 - 8
	Specifying Storage Parameters for Rollback Segments	9 - 8
	Setting Storage Parameters During Rollback Segment Creation	9 - 8
	Changing Rollback Segment Storage Parameters	9 - 9
	Privileges Required to Alter Rollback Segments	9 - 10
	Taking Rollback Segments Online and Offline	9 - 10
	Bringing Rollback Segments Online	9 - 10
	Taking Rollback Segments Offline	9 - 12
	Checking Whether a Rollback Segment Went Offline	9 - 13
	Taking Public and Private Rollback Segments Offline	9 - 13
	Privileges Required to Take Rollback Segments Online and Offline	9 - 13
	Explicitly Assigning a Transaction to a Rollback Segment	9 - 14
	Privileges Required to Assign a Transaction to a Rollback Segment	9 - 14
	Dropping Rollback Segments	9 - 15
	Privileges Required to Drop Rollback Segments	9 - 16
	Monitoring Rollback Segment Information	9 - 16
	Displaying Rollback Segment Information	9 - 16

PART IV

DATABASE SECURITY

Chapter 10

Establishing Security Policies	10 - 1
The Security Administrator	10 - 2
System Security Policy	10 - 2
Management of Database Users	10 - 2
User Authentication	10 - 2
Operating System Security	10 - 3
Data Security Policy	10 - 3
User Security Policy	10 - 4
General User Security	10 - 4
Password Security	10 - 4
Privilege Management	10 - 4
End-User Security	10 - 4
Using Roles for End-User Privilege Management	10 - 4
Administrator Security	10 - 6
Protection for Connections as SYS and SYSTEM	10 - 6
Protection for Connections as INTERNAL	10 - 6
Using Roles for Administrator Privilege Management	10 - 7
Application Developer Security	10 - 8
Application Developers and Their Privileges	10 - 8
The Application Developer's Environment Test and Production Databases	10 - 8
Free vs. Controlled Application Development	10 - 8
Roles and Privileges for Application Developers	10 - 9
Space Restrictions Imposed on Application Developers	10-10
Application Administrator Security	10-10
Auditing Policy	10-10

Chapter 11	Managing Users and Resources	11-1
	Managing Users	11-2
	Using Session and User Licensing	11-2
	Using Concurrent Usage Licensing	11-2
	Using Named User Limits	11-4
	Viewing Licensing Limits and Current Values	11-5
	Authenticating Users	11-6
	Using Operating System Authentication	11-6
	Using Database Authentication	11-7
	Creating Users	11-7
	Specifying a Name	11-8
	Setting a User's Authentication	11-9
	Assigning a Default Tablespace	11-9
	Assigning a Temporary Tablespace	11-10
	Assigning Tablespace Quotas	11-10
	Setting Default Roles	11-11
	Privileges Required to Create Users	11-11
	Altering Users	11-11
	Changing a User's Password, for Non-DBAs	11-12
	Changing a User's Default Roles	11-13
	Privileges Required to Alter Users	11-14
	Dropping Users	11-14
	Privileges Required to Drop Users	11-15
	Managing Resources with Profiles	11-16
	Creating Profiles	11-16
	Privileges Required to Create Profiles	11-17
	Using the Default Profile	11-17
	Assigning Profiles	11-17
	Altering Profiles	11-17
	Privileges Required to Alter Profiles	11-18
	Using Composite Limits	11-18
	Privileges Required to Set Resource Costs	11-20

Dropping Profiles	11-20
Privileges Required to Drop Profiles	11-21
Enabling and Disabling Resource Limits	11-21
Enabling and Disabling Resource Limits Before Startup	11-21
Enabling and Disabling Resource Limits While the Database Is Open	11-22
Listing Information About Database Users and Profiles	11-22
Examples	11-23
Chapter 12	
Controlling User Privileges and Roles	12-1
Understanding Privileges	12-2
System Privileges	12-2
Object Privileges	12-8
Granting INSERT Privileges for Columns	12-8
Statements Permitted by Each Privilege	12-9
Object Privilege Shortcut	12-9
Managing Roles	12-10
Creating Roles	12-10
Role Names in Multi-Byte Character Sets	12-10
Role Authorization	12-11
Using Pre-Defined Roles	12-12
Privileges Required to Create Roles	12-12
Changing Role Authorization	12-12
Privileges Required to Change Role Authorization	12-13
Changing a User's Default Roles	12-13
The MAX_ENABLED_ROLES Parameter	12-13
Dropping Roles	12-14
Privileges Required to Drop Roles	12-14
Granting and Revoking Privileges and Roles	12-15
Granting System Privileges and Roles	12-15
The ADMIN Option	12-15
Privileges Required to Grant System Privileges or Roles	12-16

Granting Object Privileges	12-16
The GRANT OPTION	12-17
Privileges Required to Grant Object Privileges	12-17
Revoking System Privileges and Roles	12-17
Privileges Required to Revoke System Privileges and Roles	12-18
Revoking Object Privileges	12-18
Revoking Column Selective Object Privileges	12-19
Revoking the REFERENCES Object Privilege	12-19
Privileges Required to Revoke Object Privileges	12-19
Cascading Effects of Revoking Privileges	12-19
System Privileges	12-20
Object Privileges	12-20
Granting to and Revoking from the User Group PUBLIC	12-21
When Do Grants and Revokes Take Effect?	12-21
Granting Database Roles Using the Operating System	12-22
Using Operating System Role Identification	12-22
Granting and Revoking Roles When OS_ROLES=TRUE	12-23
Enabling and Disabling Roles When OS_ROLES=TRUE	12-24
Using the Multi-Threaded Server with Operating System Role Management	12-24
Listing Privilege and Role Information	12-25
Examples	12-25
Chapter 13	
Auditing Database Use	13-1
Guidelines for Auditing	13-2
Choose Between Auditing with the Database and Auditing with the Operating System	13-2
Keep Audited Information Manageable	13-2
Auditing Suspicious Database Activity	13-3
Auditing Normal Database Activity	13-3
Creating and Deleting the Database Audit Trail Views	13-4
Creating the Audit Trail Views	13-4

Deleting the Audit Trail Views	13-5
Setting Auditing Options	13-5
Statement Audit Options	13-5
Shortcuts for Statement Audit Options	13-7
Auditing Connections and Disconnections	13-7
Additional Statement Options	13-8
Privilege Audit Options	13-8
Object Audit Options	13-9
Shortcut for Object Audit Options	13-10
Enabling Audit Options	13-10
Enabling Statement and Privilege Auditing	13-11
Enabling Object Auditing	13-11
Disabling Audit Options	13-12
Disabling Statement and Privilege Auditing	13-12
Disabling Object Auditing	13-13
Enabling and Disabling Database Auditing	13-14
Controlling the Growth and Size of the Audit Trail	13-14
Purging Audit Records from the Audit Trail	13-15
Reducing the Size of the Audit Trail	13-16
Protecting the Audit Trail	13-17
Viewing Database Audit Trail Information	13-17
Auditing Through Database Triggers	13-20

PART V**DISTRIBUTED PROCESSING AND
DISTRIBUTED DATABASES**

Chapter 14	Administering Client-server Systems	14-1
	Deciding Where to Locate Client Applications	14-2
	Developing Distributed Database Applications	14-2
	Connecting Between ORACLE Server Versions	14-3
Chapter 15	Managing Distributed Databases	15-1
	Guidelines for Implementing a Distributed Database System	15-2
	Assign Distributed Database Administrators	15-2
	Consider Network and Communication Issues	15-2
	Define Global Database Names and Database Links	15-2
	Decide Where to Locate the Data	15-2
	Enforce Security in a Distributed Database System	15-3
	Plan Distributed Database Backup and Recovery	15-3
	Managing Global Database Names	15-3
	Choosing a Global Database Name	15-3
	Creating a Global Database Name	15-4
	Changing a Database's Global Name	15-5
	Enforcing Global Naming	15-6
	Examining Global Database Names	15-6
	Managing Database Links	15-7
	Guidelines for Database Links	15-7
	Creating Database Links	15-7
	Privileges Required to Create a Database Link	15-9
	Controlling Connections Established by the RECO Background Process	15-9
	Dropping Database Links	15-9
	Privileges Required to Drop a Database Link	15-10
	Examining Available Database Links	15-10
	Limiting the Number of Active Database Links	15-10
	Implementing Location Transparency	15-11

Object Resolution and Location Transparency	15-11
Circular References and Location Transparency	15-12
Balancing Location Transparency and Security	15-12
Privilege Management With Views	15-12
Privilege Management With Procedures	15-13
Privilege Management With Synonyms	15-13
Limiting the Number of Distributed Transactions Per Node	15-14
Setting the Commit Point Strength of an Instance	15-15
Troubleshooting Distributed Transaction Problems	15-15
Failures that Interrupt a Two-Phase Commit	15-15
Failures that Prevent Access of Data	15-16
Manually Overriding In-Doubt Transactions	15-17
Manual Override Example	15-19
Manually Committing In-Doubt Transactions	15-25
Manually Rolling Back In-Doubt Transactions	15-26
Privileges Required to Manually Commit or Rollback In-Doubt Transactions	15-26
Changing Connection Hold Time	15-27
Setting a Limit on Distributed Transactions	15-27
Testing Distributed Transaction Recovery Features	15-27
Forcing a Distributed Transaction to Fail	15-27
Disabling and Enabling RECO	15-28

Chapter 16	Managing Table Snapshots	16 - 1
	Managing Snapshots	16 - 2
	Preparing to Create Snapshots	16 - 2
	Creating Snapshots	16 - 2
	Naming Snapshots	16 - 3
	Creating a Clustered Snapshot	16 - 3
	Privileges Required to Create Snapshots	16 - 3
	Understanding the Internals of Snapshot Creation	16 - 4
	Setting Storage Parameters for Snapshots	16 - 4
	Privileges Required to Alter a Snapshot	16 - 5
	Refreshing Snapshots	16 - 5
	Configuring Automatic Refreshes	16-5
	Manually Refreshing Snapshots	16-7
	Refreshing Snapshots and Referential Integrity	16-8
	Declarative Constraints and Snapshots	16-9
	Indexing Snapshots	16-9
	Operations on a Master Table that Affect Snapshots	16-9
	Snapshots and Media Failure	16-10
	Dropping Snapshots	16-10
	Privileges Required to Drop a Snapshot	16-10
	Managing Snapshot Logs	16-10
	Creation Order of a Simple Snapshot and the Snapshot Log	16-10
	Creating a Snapshot Log	16-11
	Naming Snapshot Logs	16-12
	The Internals of Snapshot Log Creation	16-12
	Privileges Required to Create Snapshot Logs	16-12
	Setting Storage Parameters for Snapshot Logs	16-13
	Altering Snapshot Log Storage Parameters	16-13
	Managing Snapshot Log Space Use	16-13
	Reducing Space Allocation for a Snapshot Log	16-15
	Dropping Snapshot Logs	16-15

	Privileges Required to Drop a Snapshot Log	16-16
	Manually Copying Tables Among Nodes of a Distributed Database	16-16
PART VI	DATABASE BACKUP AND RECOVERY	
Chapter 17	Archiving Redo Information	17-1
	Choosing Between NOARCHIVELOG and ARCHIVELOG Mode	17-2
	Turning Archiving On and Off	17-3
	Setting the Initial Database Archiving Mode at Database Creation	17-3
	Changing the Database Archiving Mode After Database Creation	17-3
	Enabling Automatic Archiving	17-4
	Enabling Automatic Archiving at Instance Startup	17-5
	Enabling Automatic Archiving After Instance Startup	17-5
	Disabling Automatic Archiving	17-6
	Disabling Automatic Archiving at Instance Startup	17-6
	Disabling Automatic Archiving after Instance Startup	17-6
	Performing Manual Archiving	17-6
	Privileges Required to Manually Archive an Online Redo Log Group	17-8
	Tuning Archiving	17-8
	Minimizing the Impact on System Performance	17-8
	Increasing Archiving Speed	17-9
	Displaying Archiving Status Information	17-9
	Privileges Required to List Archive Status	17-10
	Specifying the Archived Redo Log Filename Format and Destination	17-10

Chapter 18	Backing Up a Database	18-1
	Guidelines for Database Backups	18-2
	Plan Backup and Recovery Strategies Before Database Creation	18-2
	Test Backup and Recovery Strategies	18-2
	Perform Operating System Backups Frequently and Regularly	18-2
	Appropriately Backup the Database Before and After Modifying the Database Structure	18-2
	Back Up Often-Used Tablespaces Frequently	18-3
	Keep Older Backups	18-3
	Export Database Data for Added Protection and Flexibility	18-3
	Consider Distributed Database Backups	18-4
	Creating a Backup Strategy	18-4
	Backup Strategies in NOARCHIVELOG Mode	18-5
	Backup Strategies in ARCHIVELOG Mode	18-5
	Taking Backups	18-6
	Listing Database Files Before Backup	18-6
	Performing Full Backups	18-7
	Preparing to Take a Full Backup	18-7
	Steps for a Full Backup	18-8
	Performing Partial Backups	18-8
	Online Tablespace and Online Data File Backups	18-9
	Offline Tablespace and Offline Data File Backups	18-13
	Control File Backups	18-15
	Backing Up the Control File to the Trace File	18-15
	Privileges Required to Backup Control Files	18-17
	Recovering From a Failure During a Backup	18-17
	Using the Export and Import Utilities for Supplemental Database Protection	18-17
	Using Export	18-17
	Incremental Export	18-18
	Cumulative Exports	18-18

	Complete Exports	18-18
	Using Import	18-18
Chapter 19	Recovering a Database	19-1
	Guidelines and Prerequisites for Recovering from a Media Failure	19-2
	Test Backup and Recovery Strategies	19-2
	Decide What Type of Recovery Operation Is Appropriate	19-2
	Correct or Avoid a Disk Problem	19-3
	Coordinate Distributed Recovery	19-3
	Coordinate Time-Based and Change-Based Distributed Database Recovery	19-4
	Recover Databases with Snapshots	19-5
	Performing Media Recovery	19-5
	Issues Common to All Media Recovery Operations	19-5
	Using Messages that Indicate Media Failures	19-6
	Determining Whether a File Needs Recovery	19-6
	Deciding Which Recovery Action to Perform	19-6
	Restoring Damaged Data Files	19-8
	Restoring Necessary Archived Redo Log Files	19-8
	Starting Media Recovery	19-9
	Applying Redo Log Files	19-12
	Interrupting Media Recovery	19-16
	Restoring a Full Backup	19-16
	Complete Media Recovery	19-17
	Performing Closed Database Recovery	19-18
	Performing Open Database-Offline Tablespace Recovery	19-20
	Incomplete Media Recovery	19-22
	Performing Cancel-Based, Time-Based, or Change-Based Recovery	19-23
	Examples of Media Failures and Appropriate Recovery Procedures	19-27
	Types of Media Failures	19-27

Loss of Data Files	19-28
Loss of Data Files, NOARCHIVELOG Mode	19-28
Loss of Data Files, ARCHIVELOG Mode	19-28
Loss of Online Redo Log Files	19-29
Loss of Online Redo Log Members or Mirrored Online Redo Log	19-29
Loss of All Online Redo Log Members of an Online Redo Log Group	19-30
Loss of Archived Redo Log Files	19-34
Loss of Control Files	19-34
Loss of Mirrored Control Files	19-35
Loss of Non-Mirrored Control Files	19-35
Recovery From User Errors	19-36

PART VII

DATABASE AND INSTANCE TUNING

Chapter 20

Understanding the Tuning Process	20-1
Setting Goals for Tuning	20-2
Steps of the Tuning Process	20-2
Before Beginning the Process	20-3
Step One: Tuning Memory Allocation	20-3
Step Two: Tuning I/O	20-3
Step Three: Tuning Contention	20-3
After Completing the Process	20-4
Top Tuning Tips	20-4

Chapter 21	Tuning Memory Allocation	21 - 1
	The Importance of Memory Allocation	21 - 2
	Steps for Tuning Memory Allocation	21 - 3
	Tuning Your Operating System	21 - 3
	Tuning Private SQL and PL/SQL Areas	21 - 3
	Tuning the Shared Pool	21 - 3
	Tuning the Buffer Cache	21 - 3
	Tuning Your Operating System	21 - 4
	Reducing Paging and Swapping	21 - 4
	Tuning the System Global Area (SGA)	21 - 5
	User Memory Allocation	21 - 5
	Tuning Private SQL and PL/SQL Areas	21 - 6
	Identifying Unnecessary Parse Calls	21 - 6
	Reducing Unnecessary Parse Calls	21 - 6
	Tuning the Shared Pool	21 - 8
	Tuning the Library Cache	21 - 8
	Examining Library Cache Activity	21 - 8
	Reducing Library Cache Misses	21 - 10
	Speeding Access to Shared SQL Areas on Execution Calls	21 - 13
	Tuning the Data Dictionary Cache	21 - 14
	Examining Data Dictionary Cache Activity	21 - 14
	Reducing Data Dictionary Cache Misses	21 - 15
	Tuning the Shared Pool with the Multi-Threaded Server	21 - 15
	The V\$SESSTAT Table	21 - 16
	Querying the V\$SESSTAT Table	21 - 16
	Interpreting the V\$SESSTAT Table	21 - 16
	Tuning the Buffer Cache	21 - 17
	Examining Buffer Cache Activity	21 - 17
	Reducing Buffer Cache Misses	21 - 18
	The X\$KCBRBH Table	21 - 18
	Enabling the X\$KCBRBH Table	21 - 18
	Querying the X\$KCBRBH Table	21 - 19

	Grouping Rows in the X\$KCBRBH Table	21-19
	Removing Unnecessary Buffers	21-21
	The X\$KCBCBH Table	21-21
	Enabling the X\$KCBCBH Table	21-21
	Querying the X\$KCBCBH Table	21-22
	Grouping Rows in the X\$KCBCBH Table	21-22
	Reallocating Memory	21-24
Chapter 22	Tuning I/O	22-1
	The Importance of Tuning I/O	22-2
	Reducing Disk Contention	22-2
	What Is Disk Contention?	22-2
	Monitoring Disk Activity	22-2
	Monitoring ORACLE Disk Activity	22-3
	Monitoring Operating System Disk Activity	22-3
	Distributing I/O	22-4
	Separating Data Files and Redo Log Files	22-4
	“Striping” Table Data	22-5
	Separating Tables and Indexes	22-6
	Eliminating Other Disk I/O	22-6
	Allocating Space in Data Blocks	22-7
	Migrated and Chained Rows	22-7
	Avoiding Dynamic Space Management	22-9
	Detecting Dynamic Extension	22-9
	Allocating Extents	22-10
	Avoiding Dynamic Space Management in Rollback Segments	22-11
	For Long Queries	22-11
	For Long Transactions	22-11
	For OLTP Transactions	22-12

Chapter 23	Tuning Contention	23 - 1
	Reducing Contention for Rollback Segments	23 - 2
	Identifying Rollback Segment Contention	23 - 2
	Creating Rollback Segments	23 - 3
	Reducing Contention for Multi-Threaded Server Processes	23 - 4
	Reducing Contention for Dispatcher Processes	23 - 4
	Identifying Contention for Dispatcher Processes	23 - 4
	Adding Dispatcher Processes	23 - 6
	Reducing Contention for Shared Server Processes	23 - 7
	Identifying Contention for Shared Server Processes	23 - 7
	Adding Shared Server Processes	23 - 8
	Reducing Contention for Redo Log Buffer Latches	23 - 9
	Space in the Redo Log Buffer	23 - 9
	Redo Log Buffer Latches	23 - 10
	The Redo Allocation Latch	23 - 10
	Redo Copy Latches	23 - 10
	Examining Redo Log Activity	23 - 11
	Reducing Latch Contention	23 - 13
	Reducing Contention for the Redo Allocation Latch	23 - 13
	Reducing Contention for Redo Copy Latches	23 - 13
	Assigning All ORACLE Processes Equal Priority	23 - 14
 Chapter 24	 Additional Tuning Considerations	 24 - 1
	Tuning Sorts	24 - 2
	Allocating Memory for Sort Areas	24 - 2
	Recognizing Large Sorts	24 - 2
	Increasing Sort Area Size	24 - 2
	Performance Benefits of Large Sort Areas	24 - 3
	Performance Tradeoffs for Large Sort Areas	24 - 3
	Avoiding Sorts	24 - 4
	The NOSORT Option	24 - 4
	Choosing When to Use the NOSORT Option	24 - 4

Reducing Free List Contention	2 4 - 5
Identifying Free List Contention	2 4 - 5
Adding More Free Lists	2 4 - 6
Tuning Checkpoints	2 4 - 6
How Checkpoints Affect Performance	2 4 - 6
Recovery Time Performance	2 4 - 6
Runtime Performance	2 4 - 6
Choosing Checkpoint Frequency	2 4 - 7
Reducing the Performance Impact of a Checkpoint	2 4 - 7

PART VI

REFERENCE

Appendix A

Initialization Parameter Files	A - 1
Specifying Values in the Parameter File	A - 2
Changing Parameter Values	A - 3
Displaying Current Parameter Values	A - 3
Groups of Parameters	A - 4
Variable Parameters	A - 4
Global Constants Parameters (with Prefix “GC”)	A - 5
Operating System Dependent Parameters	A - 5
Derived Parameters	A - 5
When Parameters Are Set Incorrectly	A - 5
Special Symbols and Keywords	A - 6
Reading the Parameter Descriptions	A - 7
Parameter Descriptions	A - 8

Appendix B	Data Dictionary Reference	B - 1
	Data Dictionary SQL Script Files	B - 2
	Standard Scripts	B - 2
	Data Dictionary Scripts	B - 2
	Procedure Scripts	B - 3
	User Scripts	B - 4
	Migration Scripts	B - 4
	Data Dictionary Views	B - 5
	The Dynamic Performance Tables	B-72
	Access to the Dynamic Performance Tables	B-73
	Granting Access to All Tables	B-73
	Granting Access on Selected Tables	B-73
	Descriptions of Individual Tables	B-73
Appendix C	National Language Support	C - 1
	Specifying Character Sets	C - 2
	Character Set Conversion	C - 2
	Specifying Language-Dependent Behavior for a Session	C - 3
	NLS_LANG Parameter	C - 3
	Specifying NLS_LANG	C - 3
	Client/Server Architecture	C - 4
	Default Character Set	C - 4
	Overriding Language and Territory Specifications	C - 4
	Specifying Default Language-Dependent Behavior	C - 5
	NLS_LANGUAGE Parameter	C - 5
	NLS_TERRITORY Parameter	C - 6
	Overriding the Standard Defaults	C - 7
	NLS_DATE_FORMAT Parameter	C - 7
	NLS_DATE_LANGUAGE Parameter	C - 8
	NLS_NUMERIC_CHARACTERS Parameter	C - 9
	NLS_CURRENCY Parameter	C - 9
	NLS_ISO_CURRENCY Parameter	C - 10

	NLS_SORT Parameter	C - 11
	Date and Number Formats	C - 11
	Date Formats	C - 11
	Week and Day Number Conventions	C - 11
	Number Formats	C - 12
Appendix D	Database Limits	D - 1
Appendix E	Operating System Specific Information	E - 1
	Index	

PART

I

BASIC DATABASE ADMINISTRATION

1

ADMINISTERING ORACLE DATABASES

This chapter discusses the responsibilities of the person who administers the ORACLE Server, the database administrator.

This chapter covers the following topics:

- database administrator responsibilities
- the administrator's operating system account and the DBA role
- initial administrator usernames of an ORACLE database
- administrator tools and utilities
- database administrator priorities
- how to identify releases of ORACLE

Database Administrator Responsibilities

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 (or DBA) is this manager.

Every database requires at least one person to perform administrative duties; if a database is large, these administrative duties can be divided among multiple administrators.

As a database administrator, your responsibilities can include:

- 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) once 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.

Responsibilities of Other Types of Users

Other types of ORACLE users have responsibilities, as well.

Application Developers

An application developer's responsibilities include:

- 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

Database Users

A typical user's responsibilities include:

- entering, modifying, and deleting data, where permitted
- generating reports of data

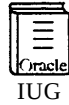
Database Administrator Security and Privileges

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. This section describes the privileges you need and database security facilities of which you should be aware.

Note: Access to a database administrator's account should be tightly controlled. See "Administrator Security" on page 10-6 for security guidelines for administrative personnel.

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 this is the case, your operating system account might require more operating system privileges or access rights than normal 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, the SQL*DBA program requires that your operating system account or ID be distinguished in some way to allow you to use *operating system privileged* SQL*DBA commands. The method of distinguishing a database administrator's account is operating system specific; see your installation and user's guide for information.

Connecting as INTERNAL

Database startup and shutdown are powerful administrative capabilities. To protect their use, only those who connect to ORACLE as INTERNAL can execute them. Depending on your operating system, you must fulfill one of the following requirements to connect to ORACLE as INTERNAL:

- Your operating system account has the operating system privileges that allow you to connect as INTERNAL.
- You are authorized to connect as INTERNAL.
- Your database has a password for INTERNAL connections, and you know the password.

In addition, you must use a dedicated server when you connect as INTERNAL. You cannot CONNECT INTERNAL through a dispatcher and shared servers.



These requirements provide an extra layer of security to prevent unauthorized users from starting up or shutting down ORACLE databases. (See your installation or user's guide for details on how to set a user's privileges for connections as INTERNAL on your system.)

Systems with passwords for INTERNAL have additional considerations, which are described in the following sections.

Using a Password for INTERNAL

Some operating systems allow you to set a password for INTERNAL connections; some even require such a password.



You can set the INTERNAL password during ORACLE Server installation, and Oracle provides a utility for managing the password (creating, changing, and removing it) with the ORACLE release. The utility is operating system specific; see your installation and users guide for details.

INTERNAL and Non-Secure Connections

If you fulfill the requirements for connecting as INTERNAL and are using a secure connection (such as a single-task connection, available on some systems), you can always connect as INTERNAL.

However, if you are using a non-secure connection (such as most network connections), you *must* use the INTERNAL password to connect as INTERNAL; this requirement implies that your system must

have an INTERNAL password to use non-secure connections for INTERNAL.



On some operating systems, you can enable and disable CONNECT INTERNAL over non-secure connections entirely. You make this choice during ORACLE installation, and can change the setting later. How this option is set is operating system specific; see your installation and user's guide for details.

Note: Some operating systems always prohibit all non-secure INTERNAL connections, even those for which you supply a password.

Alternatives to INTERNAL: OSOPER and OSDBA

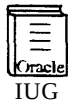
Two special roles are used to control database operations that are executed when a database is not mounted: OSOPER and OSDBA. By granting these roles to DBAs rather than granting them the privileges for connecting as INTERNAL, you can protect use of INTERNAL.

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 as INTERNAL, 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; see your installation and user's guide for information.



Note: For operating systems that do not support operating system roles, you must specify a password with INTERNAL. If the password is valid, either OSDBA or OSOPER is enabled; each has a separate password. (See "Using a Password for INTERNAL" on page 1-4.)

The DBA Role

A pre-defined role named "DBA" is automatically created with every ORACLE database. This role contains all database system privileges.

Therefore, it is very powerful and should only be granted to fully functional database administrators.

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. See “Altering Users” on page 11-11.

You will probably want to create at least one additional administrator username, to use when performing daily administrative tasks. The section “Administrator Security” on page 10-6 discusses how to establish and maintain privileges for administrative users.

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; see “Changing Storage Parameters for the Data Dictionary” on page 8-79.)

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

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

Additional tables and views that display administrative information, and internal tables and views used by ORACLE tools, are created using

the SYSTEM username. Never create tables of interest to individual users in the SYSTEM schema.

Database Administrator Utilities

Several utilities are available to help you maintain and control the ORACLE Server.

SQL*DBA

SQL*DBA allows you to monitor and control an ORACLE database. All administrative operations discussed in this book are executed using SQL*DBA.

SQL*DBA uses a superset of ANSI/ISO standard SQL commands. These commands are executed by the SQL*DBA menu-driven interface. This interface provides an intuitive approach to administering a database. The most common administrative commands are available from SQL*DBA's menu; less-often-used commands can be typed into the SQL*DBA input window and executed. You select menu commands using a pointing device (such as a mouse) or cursor keys.



SQL*DBA is documented in the *ORACLE7 Server Utilities User's Guide*.

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. SQL*Loader's use and commands are detailed in the *ORACLE7 Server Utilities User's Guide*.

Export and Import Utilities



The Export and Import utilities allow you to move existing data in ORACLE format to and from ORACLE databases. For example, export files can be used to archive database data, or move data among different ORACLE databases that run on the same or different operating systems. The Export and Import utilities are covered briefly in Chapter 18, and detailed in the *ORACLE7 Server Utilities User's Guide*.

Initial Priorities of a Database Administrator

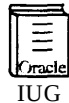
In general, you must perform a series of steps to get the database system up and running, and then maintain it. This section lists the procedures required to configure an ORACLE Server and database on any type of computer system.

1. Install the ORACLE software.
2. Evaluate the hardware of the database server.
3. Plan the database.
4. Create and start the database.
5. Implement the database design.
6. Backup the database.
7. Enroll system users.
8. Tune database performance.

The following sections explain each step in further detail.

Install the ORACLE Software

As the database administrator, you must install the ORACLE Server software and any front-end tools and database applications used to 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 SQL*Net drivers necessary to connect the remote machines to the computer that executes ORACLE.



Note: For information about identifying versions of ORACLE software, see “Identifying ORACLE Software Versions” on page 1-11.



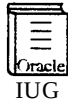
SQL*Net

Reference: Refer to the respective installation or user’s guide of your ORACLE Server, front-end tool, or SQL*Net driver for specific requirements and instructions for the installation of that product.

Evaluate the Database Server Hardware

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

- 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



Reference: Refer to your system's configuration documentation (for example, the system log or user guide).

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 whole. Be sure to plan for the growth of the database as well.

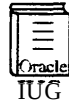
In distributed database environments, this planning stage is extremely important. The physical location of highly accessed data can dramatically affect application performance.

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.

Reference: 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 Chapters 7 through 9 for specific information on creating logical storage structures, objects, and integrity constraints for your database.

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 this is the case, all you need to do is start an instance, and mount and open the initial database.



Reference: 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 for this procedure. See Chapter 3 for database and instance startup and shutdown procedures.

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.

Reference: See Chapters 7 through 9 for instructions on creating logical storage structures and objects for your database.

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.

Reference: See Chapters 17 through 19 for instructions on customizing your backup operations and performing recovery procedures.

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.

Reference: See Chapters 10 through 12 for the procedures to create user accounts and roles, and information on complying with your license agreement.

Tune Database Performance

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

Reference: "Initial Tuning Guidelines" on page 2-12 describes steps you can take to start tuning your database immediately after creation. See Chapters 21 through 25 for more information about tuning ORACLE.

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 maybe required. For example, an ORACLE Server distribution tape might be labelled "Release 7.0.4.1." The following sections translate this number.

Version Number

The version number, such as **7**, 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 **7.0**. The maintenance release number increases when bug fixes or new features to existing programs are made available.

Patch Release Number

The patch release number identifies a specific level of the object code, such as **7.0.4**. 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 **7.0.4.1** or **7.0.4.1.3**. 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 ORACLE7:

7.0.0	the first distribution of ORACLE7 (technically a maintenance release)
7.2.0	the second maintenance release (the third release in all) of ORACLE7
7.2.2	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 ORACLE7 Server Release 7.0.12.2 system working with SQL*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 V\$VERSION, as below:

```
SQLDBA> SELECT * FROM v$version  
BANNER
```

```
-----  
ORACLE7 Server Release 7.0.20.1 - Production  
PL/SQL Release 2.0.15 - Production  
CORE V2.2.5.9.0 - Production  
TNS for VMS: Version 2.0.11.1.0 - Developer's Release  
4 rows selected.
```

This information is useful if you need to call Oracle Support.

CREATING A DATABASE

This chapter lists the steps necessary to create an ORACLE database. Topics include:

- preparing to create an ORACLE database
- creating an ORACLE database
- tuning the database initially
- creating the data dictionary
- adding structures after database creation



Trusted

If you are using Trusted ORACLE, see the *Trusted ORACLE7 Server Administrator's Guide* for additional information about creating databases in that environment.

Preparing to Create a Database

Database creation prepares several operating system files so they can work together as an ORACLE database. A database need be created only once, regardless of how many data files it has or how many instances access it. Database creation can also be used to erase information in an existing database and create a new database with the same name and physical structure.

Database creation includes the following operations:

- creating new data files or erasing data that existed in previous data files
- creating 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

A database is created by issuing a statement that includes the SQL command `CREATE DATABASE`; however, before you can execute this statement, consider the following issues:

- Plan your database tables and indexes, and estimate how much space they will require. More information on tables, indexes, and space management is given in Chapters 7 and 8.
- 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. For more information on the online and archive redo logs, see Chapters 5 and 17, respectively; for information on database backup and recovery, see Chapters 18, and 19.
- Select the database character set. You must specify the database character set when you create the database, and you cannot change it later without re-creating the database. 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. (For more information on National Language Support issues, see Appendix C.)



This chapter provides general information about database creation on all operating systems; it is relatively general. The installation or user's guide for your operating system is another important source of information about installing ORACLE and creating databases for that system. When creating a database for your database system, consistently refer to your installation or user's guide.

In addition, become familiar with information in Chapter 3, "Starting Up and Shutting Down," so that you understand the principles and options of starting up and shutting down an instance, mounting and opening a database, and using parameter files.



When creating a database using Trusted ORACLE, see the *Trusted ORACLE7 Server Administrator's Guide* for additional steps and considerations.

Creation Prerequisites

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

- the operating system privileges associated with a fully operational database administrator (See "Connecting as INTERNAL" on page 1-4.)
- 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 automatically been created 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 it with the new version of the ORACLE software. (See the *ORACLE7 Server Migration Guide* and your installation or user's guide.)



For instructions about migrating to Trusted ORACLE, see the *Trusted ORACLE7 Server Administrator's Guide*.



Creating an ORACLE Database

Carry out the following steps in sequence to create each 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*DBA and connect to ORACLE as INTERNAL.
6. Start an instance.
7. Create the database.
8. Back up the database.

The following sections describe each step.

Step 1: Back Up 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, data files, redo log files, and control files. For specific information on backing up ORACLE databases, refer to Chapter 18.

Step 2: Create Parameter Files

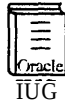
The instance (SGA and background processes) for any ORACLE database is started using a parameter file. (See “Using Parameter Files” on page 3-13.)

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. (Refer to your installation or user’s guide for more information about copying this file.)



In distributed processing environments, SQL*DBA is often executed from a client machine of the network. If a client machine is being used to execute SQL*DBA 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; see your installation or user's guide for more information about copying files among the computers of your network.

Step 3: Edit New Parameter Files

To create a new database, inspect and edit at least the following parameters of the new parameter file: DB_NAME, DB_DOMAIN, CONTROL_FILES, DB_BLOCK_SIZE, DB_BLOCK_BUFFERS, PROCESSES, and ROLLBACK_SEGMENTS.

In most cases, also edit either LICENSE_MAX_SESSIONS and LICENSE_SESSION_WARNING, or LICENSE_MAX_USERS.

Each parameter's significance is explained below. (Also see "Using Parameter Files" on page 3-13.)

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
```

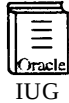
DB_NAME must be set to a text string of no more than eight characters. During database creation, the name provided for DB_NAME is recorded in the data files, redo log files, and control file of the database. If during database instance startup the value of the DB_NAME parameter (of the parameter file) 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. For more information, see "Managing Global Database Names" on page 15-3.

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. **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. This name is operating system dependent; see your installation and user's guide for details.

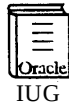
Oracle Corporation strongly recommends you use at least two control files stored on separate physical disk drives for each database. Therefore, when specifying the CONTROL_FILES parameter of the new parameter file, follow these guidelines:

- 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 8), the control files listed in the CONTROL_FILES parameter of the parameter file will be created.

DB_BLOCK_SIZE



The default data block size for every ORACLE Server is operating system specific; see your installation or user's guide. The ORACLE data block size is typically either 2K or 4K. In most cases, the default data block size should not be altered. 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. To correct for this, all databases created should have a data block size that is larger than the operating system block size.

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 data 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).

`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.

For more information on tuning the buffer cache, see the section "Tuning the Buffer Cache" beginning on page 21-17.

`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. To determine how many rollback segments you need, see Table 23-1 on page 23-3.

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.

LICENSE_MAX_SESSIONS,
LICENSE_SESSION_WARNING, and
LICENSE_MAX_USERS

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.

The following sections explain how you initially set the session and user limits. See “Using Session and User Licensing” on page 11-2 for more information about this facility, as well as instructions for managing licensing.

Setting Concurrent Usage Limits 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 used to start the instance. For example:

```
LICENSE_MAX_SESSIONS = 80
```

In addition to setting a maximum, 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.



Parallel

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 the *ORACLE7 Parallel Server Administrator's Guide* for more information about setting these limits in that environment.

Using Named User Limits 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. For example:

```
LICENSE_MAX_USERS = 200
```



Parallel

For instances running with the Parallel Server, all instances connected to the same database should have the same named user limit. See the *ORACLE7 Parallel Server Administrator's Guide* for more information on setting this limit in that environment.

Step 4: Check the Instance Identifier

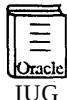


IUG

If you have other databases, check the ORACLE instance identifier.

This procedure is operating system specific; see your installation or user's guide. 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.

Step 5: Start SQL*DBA and Connect as INTERNAL



IUG

Starting SQL*DBA is operating system specific; see your installation or user's guide.

Once SQL*DBA is running, connect to the database as INTERNAL. (See "Connecting as INTERNAL" on page 1-4, as well as your installation or user's guide.)

Step 6: Start an Instance

To start an instance (SGA and background processes) to be used with the new database, use the Start Up Instance dialog box of SQL*DBA. In the Start Up Instance dialog box, make sure that the Nomount check box is selected, the database name is specified in the Database Name entry field, and the parameter file is specified in the Parameter File entry field. If you are not familiar with the Start Up Instance dialog box, see page 3-6 for an example.

After the Start Up Instance dialog box is executed, the instance is started. 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

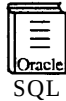
To create the new database, use the SQL command CREATE DATABASE, optionally setting parameters within the statement to name the database, establish maximum numbers of files, name files and set their sizes, and so on. When executed, a CREATE DATABASE statement performs the following operations:

- creates the data files 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 used to store data in the database
- mounts and opens the database for use

It might take a long time to allocate the specified data and redo log files and create the data dictionary.

Note: Make sure that the data files and redo log files that you specify do not conflict with files of another database. For complete information about the CREATE DATABASE command, refer to the *ORACLE7 Server SQL Language Reference Manual*. Also, for more information about character sets and database creation, see Appendix C.



Example of the CREATE DATABASE Command

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

```
CREATE DATABASE test
  LICENSE_MAX_USERS 64
  LOGFILE GROUP 1 ('test_log1a', 'test_log1b') SIZE 500K,
           GROUP 2 ('test_log2a', 'test_log2b') SIZE 500K
  DATAFILE 'test_system' SIZE 10M;
```

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 10MB data file named TEST_SYSTEM.
- The new database has two online redo log groups, each containing two 500KB members.
- The new database **does not** overwrite any existing files to create the 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 data file 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 the *ORACLE7 Server Utilities User's Guide* and the *ORACLE7 Server SQL Language Reference Manual* for more information on setting limits during database creation; see your installation or user's guide for information about operating system limits.

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 Chapter 18.

Troubleshooting Database Creation

If for any reason database creation does not succeed, complete the following steps before attempting to create it once again:

1. Shut down the instance.
2. Delete any files created by the CREATE DATABASE statement.

After completing these two steps and correcting the error that caused the failure of the database creation, continue from Step 6.

After Database Creation



After a database is created, the instance is left running and the database is open and available for normal database use. To subsequently start and stop the database, use SQL*DBA. If more than one database exists in your database system, specify the parameter file to use with any subsequent database startup. Refer to the *ORACLE7 Server Utilities User's Guide* for information about the Parameter File entry field of the Start Up Instance dialog box.

If you plan to install other ORACLE products to work with this database, refer to the installation instructions for those products; some products require you to create additional data dictionary tables. Refer to the installation or user's guide and additional 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 "Database Administrator Usernames" on

page 1-6 for more information about these usernames, and “Altering Users” on page 11-11 for the procedure to change a user’s password.

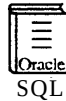
Initial Tuning Guidelines

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 these installation issues:

- allocating rollback segments
- distributing I/O across disks

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. Table 23-1 on page 23-3 shows 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 command. For more information on this command, see the *ORACLE7 Server SQL Language Reference Manual*.

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. For information on choosing sizes for your rollback segments, see the section “Reducing Contention for Rollback Segments” beginning on page 23-2.

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
- data file placement
- separation of tables and indexes

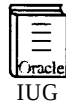
- density of data (rows per data block)

For complete information on each of these issues, see the section “Distributing I/O” beginning on page 22-4.

Creating the Data Dictionary

The data dictionary is automatically created when a database is created. Thereafter, whenever the database is in operation, ORACLE updates the data dictionary in response to every DDL statement.

The data dictionary base tables must be the first objects created in any database, because they must be present to store information about all other user-defined objects to be created. During database creation, the initialization parameter `INIT_SQL_FILES` is read to find the names of script files that should be run immediately following database creation, to create the data dictionary and other important structures. The default filenames for this parameter vary by operating system (see your installation or user’s guide).



In general, these initial SQL files do the following tasks:

1. Define the SYSTEM tablespace and SYSTEM rollback segment.
2. Define the data dictionary base tables in the SYSTEM tablespace. (The data dictionary is created in and must remain in the SYSTEM tablespace.) For every table and column in a base table, a comment is also loaded to provide online documentation.
3. Load data into some data dictionary tables.

Table 2-1 lists the scripts that are required for the ORACLE Server with the indicated options. The appropriate scripts for your ORACLE Server options are run automatically when you create a database. They are described here because you might need to run them again, when upgrading to a new release of ORACLE; your release notes indicate when this is necessary. The names and locations of these scripts are operating system-dependent; see your installation or user’s guide. To run these scripts, you should be connected as the user SYS. The scripts whose names start with DBMS are documented more fully in the *ORACLE7 Server Application Developer’s Guide*, and in comments in the scripts themselves.



TABLE 2-1
Required SQL
Scripts

Script Name	Requires Options	Description
CATALOG.SQL	none	Creates the data dictionary and public synonyms for many of its views, and grants PUBLIC access to the synonyms; also calls the scripts CATAUDIT.SQL, CATEXP.SQL, and CATLDR.SQL
CATAUDIT.SQL	none	Creates the database audit trail and views (This is run automatically by CATALOG.SQL; it can be removed by running CATNOAUD.SQL.)
CATEXP.SQL	none	Creates data dictionary tables for Import/ Export (This is run automatically by CATALOG. SQL.)
CATLDR.SQL	none	Creates views for using SQL*Loader (This is run automatically by CATALOG.SQL.)
CATPARR.SQL	Parallel Server	Creates data dictionary views on Parallel Server information
CATTRUST.SQL	Trusted ORACLE	Defines structures needed for the Trusted ORACLE Server
CATPROC.SQL	procedural	Runs all scripts required for or used within the procedural option: CATPRC.SQL, CATSNAP.SQL, CATRPC.SQL, STANDARD.SQL, DBMSSTDY.SQL, PIPDL.SQL, PIDIAN.SQL, DIUTIL.SQL, PISTUB.SQL, DBMSUTIL.SQL, DBMSSNAP.SQL, DBMSLOCK.SQL, DBMSPIPE.SQL, DBMSALRT.SQL, DBMSOTPT.SQL, DBMSDESC.SQL
CATPRC.SQL	procedural	Creates data dictionary views for stored procedures, packages; and database triggers (This is run automatically by CATPROC.SQL.)
CATSNAP.SQL	procedural, distributed	Creates data dictionary structures for storing and maintaining snapshots (This is run automatically by CATPROC.SQL; it requires CATPRC.SQL.)
CATRPC.SQL	procedural, distributed	Creates data dictionary views on distributed database information (This is run automatically by CATPROC.SQL; it requires CATPRC.SQL.)
STANDARD. SQL	procedural	Creates PL/SQL packages for the procedural option (This is run automatically by CATPROC.SQL; it requires CATPRC.SQL.)
DBMSSTDY.SQL	procedural	Includes extensions to the package standard (This is run automatically by CATPROC.SQL; it requires STANDARD.SQL.)
PIPIDL.SQL	procedural	Creates PL/SQL packages for the procedural option (This is run automatically by CATPROC.SQL; it requires DBMSSTDY.SQL.)

Script Name	Requires Options	Description
PIDIAN.SQL	procedural	Creates PL/SQL packages for the procedural option (This is run automatically by CATPROC.SQL; it requires DBMSSTDY.SQL.)
DIUTIL.SQL	procedural	Creates PL/SQL packages for the procedural option (This is run automatically by CATPROC.SQL; it requires PIDIAN.SQL.)
PISTUB.SQL	procedural	Creates PL/SQL packages for the procedural option (This is run automatically by CATPROC.SQL; it requires DIUTIL.SQL.)
DBMSUTIL.SQL	procedural	Creates utilities that can be called from within procedures (This is run automatically by CATPROC.SQL; it requires PISTUB.SQL.)
DBMSSNAP.SQL	procedural, distributed	Creates procedures for administering snapshots (This is run automatically by CATPROC.SQL; it requires CATSNAP.SQL, and you must run it on both the snapshot and master table nodes.)
DBMSALRT.SQL	procedural	Allows users and applications to use event alerters (This is run automatically by CATPROC.SQL; it requires PISTUB.SQL.)
DBMSLOCK.SQL	procedural	Allows users and applications to use the lock package (This is run automatically by CATPROC.SQL; it requires PISTUB.SQL.)
DBMSMAIL.SQL	procedural	Allows users and applications to send Oracle*Mail messages (This is run automatically by CATPROC.SQL; it requires PISTUB.SQL. You must run it on the sending database, and run UTLMAIL.SQL on the receiving Oracle*Mail database)
DBMSOTPT.SQL	procedural	Allows application developers to receive I/O from procedures (This is run automatically by CATPROC.SQL; it requires PISTUB.SQL.)
DBMSPIPE.SQL	procedural	Allows sessions in the same instance to communicate with each other (This is run automatically by CATPROC.SQL; it requires PISTUB.SQL.)
DBMSDESC.SQL	procedural	Creates a package that allows you to describe the arguments and return values of program units (This is run automatically by CATPROC.SQL; it requires PISTUB.SQL.)

The parameter INIT_SQL_FILES can also be used to specify other files to be run during database creation, after the data dictionary is created, to create site-specific tables. For example, you could add names of your files after the default filenames:

INIT_SQL_FILES = (CATALOG.SQL, CATPROC.SQL ACME_DBA.SQL)

In this example, ACME_DBA.SQL is an additional file to run during database creation.

Creating Additional Data Dictionary Structures

Oracle supplies a number of other scripts with the ORACLE Server that create additional structures you can use in managing your database and creating database applications. These scripts are listed below; for more information on them, see Appendix B. The exact names and locations of these scripts are operating system dependent; see your installation or user's guide for more details.

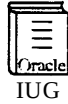


TABLE 2-2
Additional SQL
Scripts

Script Name	Options Required	Run By	Description
CATALOG6.SQL	none	SYS	Creates Version 6 data dictionary views
CATBLOCK.SQL	none	SYS	Creates the view BLOCKING_LOCKS, which shows which locks are blocking the system
CATDBSYN.SQL	none	user with access to data dictionary tables	Creates private synonyms for DBA_ views in the data dictionary
CATEXP6.SQL	none	SYS	Creates views for using the Version 6 Export utility with ORACLE7
CATNOAUD.SQL	none	SYS	Removes the database audit trail created by CATAUDIT.SQL, including its data and views
CATNOPRC.SQL	none	SYS	Removes data dictionary structures that were created by CATPRC.SQL
CATSTAT.SQL	none	SYS	Executes the ANALYZE command on structures in the data dictionary, to allow cost-based optimization of internal SQL statements; should be run periodically to update the statistics (intended for sites that do not have the procedural option; procedural option sites can instead call ANALYZE SCHEMA())
UTLBSTAT.SQL	none	any user	Begins collecting performance tuning statistics (end with UTLESTAT.SQL.)
UTLCHAIN.SQL	none	any user	Creates tables for storing the output of the ANALYZE command with CHAINED ROWS option

Script Name	Options Required	Run By	Description
UTLDTREE.SQL	procedural	any user	Creates tables and views that show dependencies between objects
UTLESTAT.SQL	none	any user	Ends collecting of performance tuning statistics started by UTLBSTAT.SQL
UTLEXCPT.SQL	none	any user	Creates the default table (EXCEPTIONS) for storing exceptions from enabling constraints
UTLEXP6.SQL	none	SYS	Returns a list of objects not exported by SQL*Net export of a Version 6 database
UTLDIDX.SQL	none	any user	Displays the results of running the scripts UTLIDXSS.SQL and UTLIDXSO.SQL
UTLOIDX.SQL	none	any user	Runs UTLIDXSS.SQL on multiple columns
UTLLOCKT.SQL	none	SYS	Displays a lock wait-for graph, in tree structure format
UTLMAIL.SQL	none	SYS	Allows Oracle*Mail database to receive messages from procedures (run on Oracle*Mail database; requires that you run DBMSMAIL.SQL on the database that will send messages)
UTLMONTR.SQL	none	SYS	Grants access to all performance tables used by SQL*DBA Monitors to PUBLIC group, including access to Monitors
UTLSAMPL.SQL	none	any user	Creates sample tables (such as EMP and DEPT) and users (such as SCOTT)
UTLSIDX.SQL	none	any user	Computes the selectivity of a column, and tests whether an index created on the column would be appropriate
UTLXPLAN.SQL	none	any user	Creates the table PLANTABLE, which holds output from the EXPLAIN PLAN command

STARTING UP AND SHUTTING DOWN

This chapter explains the procedures involved in starting and stopping an ORACLE database. Topics include:

- starting an instance and database
- changing the availability of a database
- shutting down an instance and database
- using parameter files



Trusted

If you are using Trusted ORACLE, see the *Trusted ORACLE7 Server Administrator's Guide* for information on starting up and shutting down in that environment.

General Startup Procedures

To startup a database or instance, use either the Start Up Instance dialog box or the STARTUP command (after you connect to ORACLE as INTERNAL). 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 DBAs only)

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.

The following sections give instructions for each type of startup. Examples of starting databases and instances are provided in “Examples of Database and Instance Startup” on page 3-6.

Note: You cannot start a database instance if you are connected to the database via a multi-threaded server process. (See Chapter 4 for more information about multi-threaded server processes.) Also, special conditions of database startup apply when using Trusted ORACLE; see the *Trusted ORACLE7 Server Administrator's Guide* for this information.



Preparing to Start an Instance

Before you can start an instance, you must connect as INTERNAL; you might also want to indicate for which database you are starting an instance and specify a parameter file.

Starting SQL*DBA and Connecting as INTERNAL

To startup a database or instance, you must start SQL*DBA. See the *ORACLE7 Server Utilities User's Guide* for instructions.



You must also be connected as INTERNAL. This condition applies whether you are using SQL*DBA's graphical interface or SQL commands. (See “Connecting as INTERNAL” on page 1-4.)

Specifying a Database Name

When starting a database instance, specify the name of the database that will be mounted to the instance by either:

- using the Start Up Instance dialog box, entering the database name in the Database Name entry field
- using the STARTUP command and specifying the database name

Specifying a Parameter Filename

When starting a database instance, choose a parameter file to initialize the instance's settings by either:

- using the Start Up Instance dialog box and entering a filename in the Parameter File entry field
- using the STARTUP command with the PFILE option and a fully specified filename



The specification of filenames is operating system specific; see your installation or user's guide. If no filename is entered, ORACLE uses the default filename.

Starting an Instance Without Mounting a Database

You might want to start an instance without mounting a database; this is usually the case only during database creation. To do this, use one of the following options of SQL*DBA:

- the Start Up Instance dialog box, selecting the Nomount radio button
- the STARTUP command with the NOMOUNT option

Starting an Instance and Mounting a Database

You might want to start an instance and mount a database, but not open the database, to perform specific maintenance operations. For example, the database must be mounted but not open during the following tasks:

- renaming data files
- 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 using one of the following options of SQL*DBA:

- the Start Up Instance dialog box, selecting the Mount radio button
- the STARTUP command with the MOUNT option

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 stage allows any valid user to connect to the database and perform typical data access operations.

Start an instance, and mount and open the database, using one of the following options of SQL*DBA:

- the Start Up Instance dialog box, selecting the Open radio button
- the STARTUP command with the OPEN option

Restricting Access to a Database at Startup

You might want to start an instance, and mount and open a database in restricted mode so that the database is available only to administrative personnel and not to typical database users. Use this mode of database startup when you need to accomplish one of these 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

Normally, 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 using one of the following options of SQL*DBA:

- the Start Up Instance dialog box, selecting the Restrict to DBAs check box
- the STARTUP command with the RESTRICT option

Note: Subsequently, you can make the database accessible to users without the RESTRICTED SESSION system privilege; see “Restricting Access to an Open Database” on page 3-9 for more information.

Forcing an Instance to Start

In unusual circumstances, you might experience problems when attempting to start a database instance. A database instance should not be forced to start unless:

- The current instance cannot be successfully shut down using either the Normal or Immediate options of the Shut Down menu (or an equivalent SHUTDOWN statement).
- You experience problems when starting an instance.

Note: Aborting the current instance has particular side effects; see “Aborting an Instance” on page 3-11.

If such a situation arises, you can usually solve the problem by starting a new instance (and optionally mounting and opening the database) using either of the following options of SQL*DBA:

- the Start Up Instance dialog box with the Force check box
- the STARTUP command with the FORCE option

These options of the Start Up Instance dialog box and the STARTUP command effectively shut down the current instance and then start a new instance (optionally mounting and opening the database).

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 one of the following options of SQL*DBA:

- the Start Up Instance dialog box with the Recover check box
- the STARTUP command with the RECOVER option

ORACLE might need archived redo log files to perform the automatic recovery. See Chapter 19 for more information about database recovery.

Starting in Exclusive or Parallel Mode



Parallel

If your ORACLE Server allows multiple instances to access a single database concurrently, you must choose whether to mount the database exclusively or in parallel. See the *ORACLE7 Parallel Server Administrator's Guide* for more information.

Miscellaneous Startup Topics



IUG

The following topics are unique options of database and instance startup that vary depending on your operating system. See your installation or user's guide for more information about each topic.

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 doing this 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.

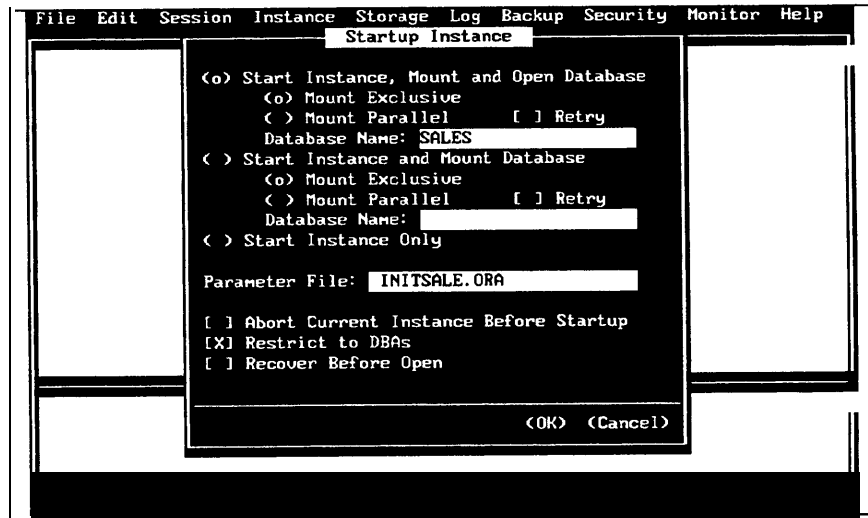
Examples of Database and Instance Startup

This section gives several examples of instance startup using the menu interface and commands of SQL*DBA. In each example, the DBA is already connected as INTERNAL.

The Startup Instance Dialog

The Startup Instance dialog allows you to start an instance and optionally mount and open a database. For example, Figure 3-1 shows the Startup Instance dialog.

FIGURE 3-1
The Start up Instance Dialog



This example results in starting an instance using the parameter file INITSALE.ORA, mounting and opening the database named SALES in exclusive mode, and restricting access to administrative personnel. (For more information about the Start Up Instance dialog, see the *ORACLE7 Server Utilities User's Guide*.)

The STARTUP Command

The command equivalent of using the Start Up Instance dialog is the SQL*DBA command STARTUP. The following statement is the command equivalent of the Start Up Instance dialog illustrated in Figure 3-1:

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



There are several restrictions to combining options of the STARTUP command. See the *ORACLE7 Server Utilities User's Guide* for more information.

Troubleshooting Startup

Sometimes you might have problems starting up an instance. For example, control files, database files, or redo log files might not be available. This section describes how to tell when this is the case and how to deal with the problem.

Unavailable Control Files

When you attempt to mount a database, if one or more of the files specified by the CONTROL.FILES parameter do not exist or cannot be opened, ORACLE returns a warning message and does not mount the database. For information about database recovery, see Chapter 19.

Unavailable Database or Redo Log Files

When attempting to open a database, if one or more of the data files or redo log files is not available or cannot be opened, ORACLE returns a warning message and does not open the database. For information about database recovery, see Chapter 19.

Altering Database Availability

You can make a database partially available to users. For example, you can open a previously mounted but closed database, so that database users can connect to and use the database.



The following sections explain how to change a database's availability. All procedures are accomplished using options of the Instance menu of SQL*DBA or the SQL command ALTER DATABASE. For complete information about SQL*DBA menu options, see the *ORACLE7 Server Utilities Guide*; for complete information about the ALTER DATABASE command, see the *ORACLE7 Server SQL Language Reference Manual*. To accomplish any of the following procedures, you must have the ALTER DATABASE system privilege.

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. This can be accomplished by starting the instance and mounting the database in one or two steps.

Note: For a list of operations that require the database to be mounted and closed and for the procedures to start an instance and mount a database in one step, see "Starting an Instance and Mounting a Database" on page 3-3.

When mounting the database, you can indicate whether to mount the database exclusively to only this instance or mount the database so that it can be mounted concurrently by other instances.

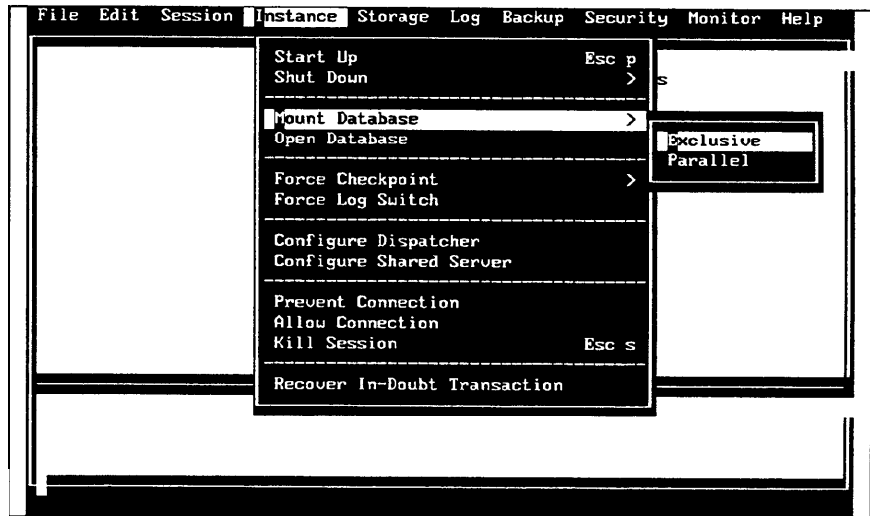
To mount a database to a previously started instance, use either of the following options:

- the Mount Database menu of SQL*DBA
- the SQL command ALTER DATABASE with the MOUNT option

The Mount Database Menu

The Mount Database menu allows you to mount a database to a previously started instance. You can mount the database in either exclusive mode or parallel mode. Figure 3-2 shows the Mount Database menu.

FIGURE 3-2
The Mount Database Menu



This example mounts a database in exclusive mode.

The command equivalent of using the Mount Database menu is the SQL command ALTER DATABASE, with the MOUNT option. (You can also select either the EXCLUSIVE or PARALLEL option if you are using the Parallel Server.) For example, the following statement is the command equivalent of the Mount Database menu illustrated in Figure 3-2:

```
ALTER DATABASE MOUNT;
```

Opening a Closed Database

You make a mounted but closed database available for general use by opening the database. To open a mounted database, use either of the following options:

- the Open option of the Instance menu of SQL*DBA
- the SQL command ALTER DATABASE with the OPEN option

For example, the following statement opens a mounted database:

```
ALTER DATABASE OPEN;
```

Once you have executed this command, any valid ORACLE user with the CREATE SESSION system privilege can connect to the database.

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 the instance. Typically, only administrators have the RESTRICTED SESSION system privilege. Restricted mode is useful when you need to perform these 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 Prevent Connections option of the Instance menu of SQL*DBA or the SQL command ALTER SYSTEM with the ENABLE RESTRICTED SESSION option. After placing an instance in restricted mode, you might also want to kill all current user sessions before performing any administrative tasks. (See “Killing Sessions” on page 413.)

To lift an instance from restricted mode, use the Allow Connections option of the Instance menu of SQL*DBA or the SQL command ALTER SYSTEM with the DISABLE RESTRICTED SESSION option.

Note: Alternatively, you can start a database instance, and mount and open the database in restricted mode. See “Restricting Access to a Database at Startup” on page 3-4.

Shutting Down a Database

To initiate database shutdown, use either the Shut Down menu or the SHUTDOWN command of SQL*DBA. 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 similar to the following message:

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

The following sections explain how to shut down a database. Examples of the Shut Down menu and the SHUTDOWN command are provided in “Examples of Database Shutdown” on page 3-12.

Note: You cannot shut down a database if you are connected to the database via a multi-threaded server process. See Chapter 4 for more information about multi-threaded server processes. Also, several special options and conditions of database shutdown that apply when using Trusted ORACLE in OS MAC mode are not discussed in this section; for more information, see the *Trusted ORACLE7 Server Administrator's Guide*.



Connecting as INTERNAL

To shut down a database and instance, you must first be connected as INTERNAL. This condition applies whether you are using SQL*DBA's graphical interface or SQL commands. (See “Connecting as INTERNAL” on page 1-4.)

Shutting Down a Database Under Normal Conditions

Normal database shutdown proceeds as follows:

- 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 either of the following options of SQL*DBA:

- the Normal option of the Shut Down menu
- the SHUTDOWN command with the NORMAL option

Shutting Down a Database Immediately

In emergency situations, you can immediately shut down a database; use immediate database shutdown only in situations such as:

- 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:

- Current client SQL statements being processed by ORACLE are terminated immediately.
- 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.
- The next startup of the database might require instance recovery (which is automatically performed by ORACLE).

To shut down a database immediately, use either of the following options of SQL*DBA:

- the Immediate option of the Shut Down menu
- the SHUTDOWN command with the IMMEDIATE option

Aborting an Instance

You can shutdown a database instantaneously by aborting the database's instance. Avoid this type of shutdown; use it 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 as follows:

- 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.
- The next STARTUP of the database will require instance recovery procedures; however, instance recovery will be automatically performed during database startup.

If the normal and immediate shutdown options both do not work, abort the current database instance immediately by using either of the following options of SQL*DBA:

- the Abort Instance option of the Shut Down menu
- the SHUTDOWN command with the ABORT option

Examples of Database Shutdown



Utilities

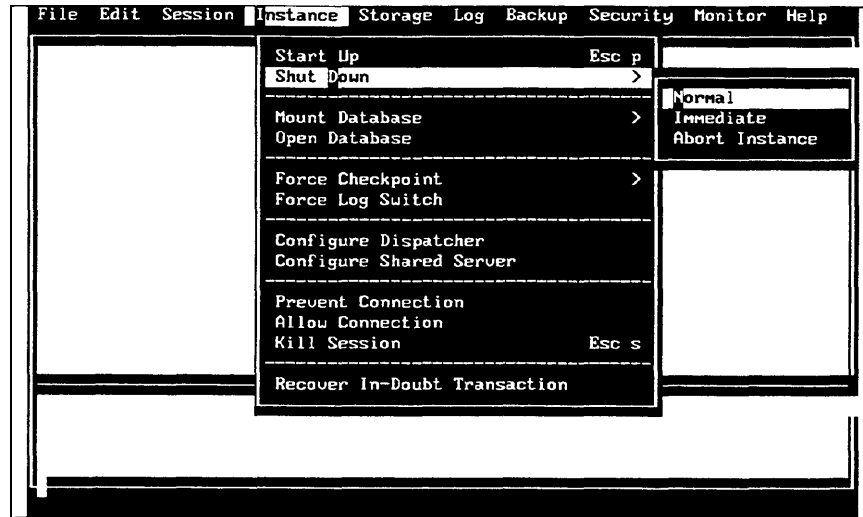
This section gives examples of database and instance shutdown using the menu interface and commands of SQL*DBA. In all of the examples, the DBA has already connected as INTERNAL.

For more information about the Shut Down menu or the SHUTDOWN command, see the *ORACLE7 Server Utilities User's Guide*.

The Shut Down Menu

The Shut Down menu shuts down a database. For example, Figure 3-3 shows the Shut Down menu.

FIGURE 3-3
The Shut Down Menu



The option selected shuts down the database under normal conditions.

The SHUTDOWN Command

The command equivalent of using the Shut Down menu is the SQL*DBA command SHUTDOWN. For example, the following statement is the command equivalent of the Shut Down menu illustrated in Figure 3-3 of the previous section:

```
SHUTDOWN NORMAL;
```

Using Parameter Files

To start an instance, ORACLE must read a *parameter file*. Often, although not always, this file is named INIT.ORA or INITsid.ORA, where *sid* is operating system specific. See your installation or user's guide for more information about this file.

The Sample Parameter File

A sample parameter file (INIT.ORA or INITsid.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 Chapters 20 through 24 for information about optimizing a database's performance using the parameter file.

Parameter File Character Set

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

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 to optimize the database's performance in different situations.

The Location of the Parameter File in Distributed Processing Environments

SQL*DBA 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 that executes SQL*DBA.

For example, in non-distributed processing installations, the same computer executes ORACLE and SQL*DBA; therefore, this computer has the parameter file stored on one of its disk drives.

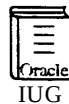
However, in distributed processing installations, local client workstations can execute SQL*DBA to administer a database stored on a remote machine. In this type of configuration, the local client machines must each store a copy of the parameter file for the corresponding databases.



Some subjects concerning the setup and implementation of SQL*DBA are operating system dependent; see your installation or user's guide.

Editing Parameter Files

Methods of editing parameter files are operating system specific. The standard way to change the parameter values in a parameter file is to edit the file with a basic text editor.



PART

II

ORACLE SERVER CONFIGURATION

4

MANAGING ORACLE PROCESSES

This chapter explains how to manage the processes of an ORACLE instance. Topics include:

- starting single-process and multiple-process ORACLE instances
- configuring and starting ORACLE with multi-threaded servers
- connecting to an instance
- monitoring ORACLE processes
- killing user processes

Starting Single-Process and Multiple-Process Instances

On most operating systems, you can start an ORACLE instance in either single-process or multiple-process mode, regardless of how ORACLE was installed or last started. If the computer that executes the ORACLE Server supports multiple processes, database instances are normally started using multiple process mode so that multiple users can concurrently access a shared database; single-process instances support only one user at a time. However, in some testing situations, you might find it useful to start an instance using a single process.

Some operating systems (such as MS-DOS) do not support multiple processes or shared memory, so a single-process instance is the only option.

Setting the Process Mode

The initialization parameter `SINGLE_PROCESS` determines which type of instance is started. To start a single process instance, set the initialization parameter `SINGLE_PROCESS` to `TRUE`; to start a multiple process instance, set `SINGLE_PROCESS` to `FALSE`. If the `SINGLE_PROCESS` parameter is not included, its default value is `FALSE`.

If your system can only run single-task ORACLE, ORACLE is automatically installed for this configuration; you have no choice but to run ORACLE in a single-task configuration.

Starting ORACLE with Only Dedicated Servers

If your system must run with dedicated server processes, ORACLE is automatically installed for this configuration. However, if your operating system can support ORACLE in this configuration, it may also support the multi-threaded server (described in the next section).

To start an instance in a dedicated server configuration, set the initialization parameters `MTS_SERVICE`, `MTS_DISPATCHERS`, `MTS_SERVERS`, and `MTS_LISTENER_ADDRESS` in a database's parameter file to null or omit them from the file altogether. (Refer to Appendix A for more information about these parameters and parameter files.)

Starting ORACLE with Multi-Threaded Servers

To setup your system in a multi-threaded server configuration, start a network listener process and set several initialization parameters:

- SHARED_POOL_SIZE
- MTS_LISTENER_ADDRESS
- MTS_SERVICE
- MTS_DISPATCHERS
- MTS_MAX_DISPATCHERS
- MTS_SERVERS
- MTS_MAX_SERVERS

You can then restart the instance, which will use the multi-threaded server configuration.



SQL*Net

Directions for managing the network listener process are given in your SQL*NET documentation; instructions for setting each parameter are given in the following sections.

Note: User processes that want to use the multi-threaded server must connect through SQL*Net, even if they are on the same machine as the ORACLE instance.

Allocating Additional Space in the Shared Pool for Shared Server

When users connect through the multi-threaded server, ORACLE needs to allocate additional space in the shared pool for storing information about the connections between the user processes, dispatchers, and servers. For each user who will connect using the multi-threaded server, add 1K to the setting of the parameter SHARED_POOL_SIZE. (For more information on this parameter, see Appendix A; also see Chapter 23 for more information on tuning.)

Setting the Listener Process Address (MTS_LISTENER_ADDRESS)

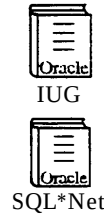
Within the database's parameter file, set the initialization parameter MTS_LISTENER_ADDRESS for each port to which the database will connect. The parameter supports the following syntax:

```
MTS_LISTENER_ADDRESS = "(addr)"
```

In this example, *addr* is an address to which the listener will listen for connection requests for a specific protocol. The file may contain several such addresses:

```
MTS_LISTENER_ADDRESS = " (ADDRESS=(PROTOCOL=tcp) (PORT=5000)\
                        (HOST=ZEUS) "
MTS_LISTENER_ADDRESS = " (ADDRESS=(PROTOCOL=decnet) (OBJECT=OUTA)\
                        (NODE=ZEUS) "
```

Note: This syntax differs slightly from that of the SQL*Net listener parameter: `MTS_LISTENER_ADDRESS` can contain only one address per line (`ADDRESS_LIST` is not supported), while the SQL*Net parameter can list multiple addresses. Otherwise, the address syntax is the same.



Each address specified in the database's parameter file must also be specified in the corresponding listener's configuration file. Addresses are specified differently for various network protocols. For more information about specifying addresses for the network listener process, refer to your installation or user's guide and SQL*Net documentation.

Specifying Service Names for Dispatchers (`MTS_SERVICE`)

Specify the name of the service associated with dispatchers using the parameter `MTS_SERVICE`. A user requests the multi-threaded server by specifying this service name in the connect string. A service name must be unique; if possible, use the instance's SID. If you do not set this parameter, it defaults to the `DB_NAME` parameter. (If `DB_NAME` is also not set, ORACLE returns the error ORA-00114, "missing value for system parameter `mts_service`," when you start the database.) To see your SID, type `SHOW LOGICAL ORA_SID` in SQL*DBA.

For example, if the dispatcher's service name is `TEST_DB`, the parameter would be set as in the following example:

```
MTS_SERVICE = "test_db"
```

Furthermore, a connect string for connecting to this dispatcher might look like the following example:

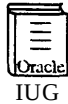
```
SQLPLUS scott/tiger@\
      (DESCRIPTION=(ADDRESS=(PROTOCOL=decnet) (NODE=hq)\
                        OBJECT=mts7))\
      (CONNECT_DATA=(SID=test_db)))
```



Refer to your installation or user's guide or SQL*Net documentation for more information about connect strings used with the multi-threaded server configuration.

Setting the Initial Number of Dispatchers

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.



The appropriate number of dispatcher processes for each instance depends upon the performance you want from your database and 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. (See your installation or user's guide for the operating system limit.) The instance must be able to provide as many connections as there are concurrent users on the database system; the more dispatchers you have, the better potential database performance users will see, since they will not have to wait as long for dispatcher service.

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:

$$\begin{array}{rcl} \text{number} & & \text{maximum number of concurrent sessions} \\ \text{of} & = & \text{CEIL (} \frac{\text{-----}}{\text{connections per dispatcher}} \text{)} \\ \text{dispatchers} & & \end{array}$$

Here, *connections/dispatcher* is operating system dependent. For example, assume that your system typically has 80 users concurrently connected via TCP/IP and 40 users connected via DECNet. In this case, the MTS_DISPATCHERS parameter should be set as follows:

```
MTS_DISPATCHERS = "TCP/IP, 3"
MTS_DISPATCHERS = "DECNET, 3"
```

When setting the MTS_DISPATCHERS parameter, you can only include protocols specified in a MTS_LISTENER_ADDRESS parameter.

After instance startup, you can start more dispatcher processes if needed; however, you can only start dispatchers that use protocols mentioned in the database's parameter file. For example, if the parameter file starts dispatchers for protocol_A and protocol_B, you cannot later start dispatchers for protocol_C without changing the parameter file and restarting the instance. (See "Adding and Removing Dispatcher Processes" on page 4-7 for more information.)

Setting the Maximum Number of Dispatchers

The parameter MTS_MAX_DISPATCHERS sets the maximum number of dispatcher processes (of all network protocols combined) that can be started for the duration of an instance. To estimate the maximum number of dispatcher processes an instance will require, use the following formula:

$$\text{MTS\_MAX\_DISPATCHERS} = \frac{\text{maximum number of concurrent sessions}}{\text{connections per dispatcher}}$$

You can create as many dispatcher processes as you need, but the total number of processes, including dispatchers, cannot exceed the host operating system's limit on the number of running processes.

Setting the Initial Number of Shared Server Processes

A number of shared server processes are started at instance startup, as determined by the parameter `MTS_SERVERS`. The appropriate number of initial shared server processes for a database system depends on how many users are typically connected and how much processing each user requires. If each user makes relatively few requests over a period of time, then each associated user process is idle for a large percentage of time. In that case, one shared server process can serve 10 to 20 users. If each user requires a significant amount of processing, a higher ratio of server processes to user processes is needed to handle requests.

If you want ORACLE to use shared servers, you must set `MTS_SERVERS` to at least 1. If you omit the parameter or set it to 0, ORACLE does not start any shared servers at all. However, you can later set `MTS_SERVERS` to a number greater than 0 while the instance is running; see "Changing the Minimum Number of Shared Server Processes" on page 4-6.

It is best to estimate fewer initial shared server processes. Additional shared servers are started automatically when needed and deallocated automatically if they remain idle for too long. However, the initial servers always remain allocated, even if they are idle. If you set the initial number of servers high, your system might incur unnecessary overhead. Experiment with the number of initial shared server processes and monitor shared servers until you find the ideal system performance for typical database activity.

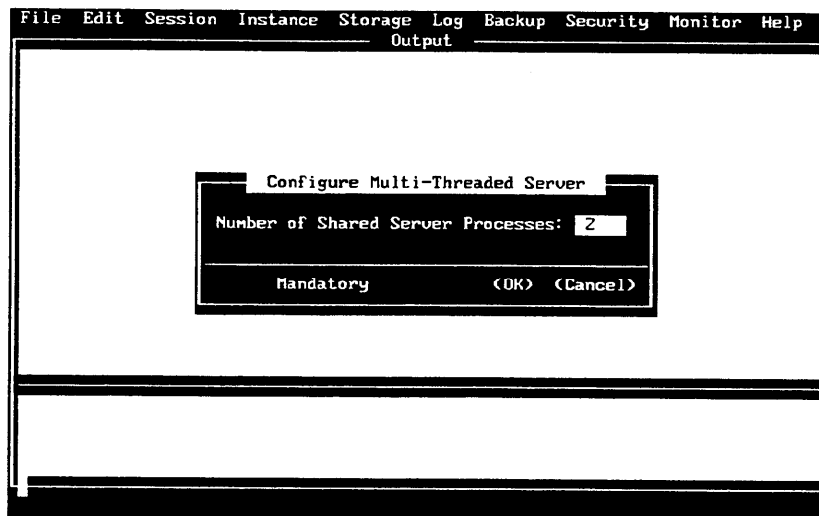
Setting the Maximum Number of Shared Server Processes

The maximum number of shared server processes that can be started for the duration of an instance is established during instance startup by the parameter `MTS_MAX_SERVERS`. In general, set this parameter to allow an appropriate number of shared server processes at times of highest activity. Experiment with this limit and monitor shared servers to determine an ideal setting for this parameter.

Changing the Minimum Number of Shared Server Processes

Once the instance has started, you can change the minimum number of shared server processes with either the Configure Servers dialog box of SQL*DBA or the SQL command `ALTER SYSTEM`. Figure 4-1 shows the Configure Multi-Threaded Server dialog.

FIGURE 4-1
The Configure Multi-Threaded
Server Dialog



This example sets the number of shared server processes to two. (ORACLE eventually kills dispatchers and servers that are idle for too long, to the minimum limit you specify.) The following statement is the command equivalent of the Configure Multi-Threaded Servers dialog box in Figure 4-1:

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

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 effectively disables the multi-threaded server temporarily.

Privileges Required to
Change the Minimum
Number of Shared Servers

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

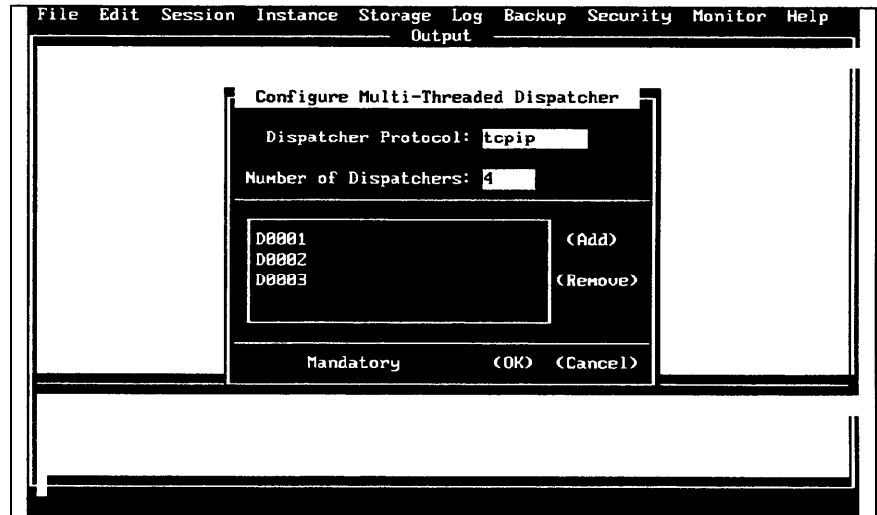
Adding and Removing Dispatcher Processes

You can control the number of dispatcher processes in the instance. If the V\$QUEUE and V\$DISPATCHER views indicate that the load on the dispatcher processes is consistently high, start additional dispatcher processes to route user requests without waiting; you may start new dispatchers until the number of dispatchers equals MTS_MAX_DISPATCHER. In contrast, if the load on dispatchers is consistently low, reduce the number of dispatchers. (See Chapter 23 for more information on tuning the multi-threaded server.)

To change the number of dispatcher processes, use the Configure Multi-Threaded Dispatchers dialog box of SQL*DBA or the ALTER

SYSTEM command. Figure 4-2 shows the Configure Multi-Threaded Dispatcher dialog:

FIGURE 4-2
The Configure Multi-Threaded Dispatcher Dialog



If the number of dispatchers was previously three, this example adds a dispatcher process. The following statement is the command equivalent of the Configure Multi-Threaded Dispatcher dialog in Figure 4-2:

```
ALTER SYSTEM
  SET MTS_DISPATCHERS = 'TCPIP,4';
```

You can start new dispatcher processes for protocols specified in the `MTS_LISTENER_ADDRESS` parameter and in the `MTS_DISPATCHERS` parameter. Therefore, you can add dispatchers only for protocols for which there are dispatchers; to start dispatchers for protocols for which there are currently no dispatchers, shutdown the database, change the parameter file, and restart the database.

If you reduce the number of dispatchers for a particular protocol, the dispatchers are not immediately removed. Rather, ORACLE eventually kills dispatchers that are idle for too long, down to the limit you specify in `MTS_DISPATCHERS`.



Changing the number of dispatchers for a specific protocol has no effect on dispatchers for other protocols. (For a complete description of the `ALTER SYSTEM` command, refer to the *ORACLE7 Server SQL Language Reference Manual*.)

Privileges Required to
Change the Number of
Dispatcher Processes

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

Deciding When to Connect to a Dedicated Server Process

If possible, users should connect to an instance via a dispatcher, to keep 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 (that is, the job can allow little or no idle time for the server process)
- to use SQL*DBA to start up, shut down, or perform media recovery on a database



SQL*Net

To request a dedicated server connection, a user must include the clause (SRVR=DEDICATED) in his or her SQL*Net TNS connect string. For a complete description of SQL*Net connect string syntax, refer to your installation or user's guide and SQL*Net documentation.

Managing ORACLE Processes

An ORACLE instance can have many background processes, which you need to track. This section explains how to identify and manage these processes.

Note: See Chapter 23 for more information on tuning ORACLE processes.

Monitoring the Processes of an ORACLE Instance

Use the Monitor feature of SQL*DBA to display current information about the processes of an ORACLE database. The following monitors of SQL*DBA are of interest:

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.

Viewing and Monitoring Locks

ORACLE also provides two facilities to display locking information for ongoing transactions within an instance:



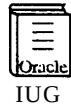
Utilities

SQL*DBA
Monitors
(Lock and Latch
Monitors)

The Monitor feature of SQL*DBA provides two monitors for displaying lock information of an instance. See the *ORACLE7 Server Utilities User's Guide* for complete information about the SQL*DBA monitors.

UTLLOCKT.SQL

The UTLLOCKT.SQL script displays a simple character lock wait-for graph in tree structured fashion. Using any ad hoc query tool (such as SQL*DBA 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 installation or user's guide. (A second script, CATBLOCK.SQL, creates the lock views that UTLLOCKT.SQL needs, so you must run it before running UTLLOCKT.SQL.)



IUG

Using Dynamic Performance Tables

Several performance tables are useful in monitoring the processes of an ORACLE instance, including:

- V\$CIRCUIT
- V\$QUEUE
- V\$DISPATCHER
- V\$SHARED_SERVER
- V\$SQLAREA
- V\$SESS_IO
- V\$LATCH
- V\$SYSSTAT

For example, the following query displays the processing load on each dispatcher process in the system:

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

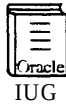
Refer to Appendix B for more information about performance tables.

Identifying ORACLE Background Processes from the Operating System

Because you might want to 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 named:

- 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



The instance identifier and the format of the ORACLE process names are operating system dependent; see your installation or user's manual.

Trace Files, the ALERT File, and Background Processes



Message

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 you as the database administrator, while other information is for Oracle support. Trace file information can also often be used in tuning applications and instances. (For more information about errors and messages, refer to the *ORACLE7 Server Messages and Codes Manual*.)

One special trace file is the *ALERT* file. The ALERT file of a database is a chronological log of messages and errors, including:

- 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, ARCHIVE LOG, and RECOVER SQL*DBA statements
- several messages and errors relating to the functions of shared server and dispatcher processes
- errors during the automatic refresh of a snapshot
- the values of all initialization parameters when 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

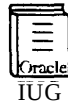
console (although many systems display information on the console). If an operation is successful, a message is written in the ALERT file as “completed” along with a timestamp.

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 is written to the LGWR trace file and the database’s ALERT file to indicate the problem. If you see such error messages, a media or I/O problem has occurred that you should correct immediately.

Also, ORACLE writes values of initialization parameters to the ALERT file, as well as other statistics of importance. For example, when you shutdown an instance normally or immediately (but 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 whether you need to upgrade your ORACLE session license. (See “Using Session and User Licensing” on page 11-2 for details.)

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); see your installation or user’s guide for more information.

Controlling the Size of Trace Files

The maximum size of all trace files (excluding the ALERT file) can be controlled 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

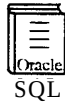
Background processes always write to a trace file when appropriate. However, trace files are written on behalf of server processes (in addition to being written to during internal errors) only if the initialization parameter `SQL_TRACE` is set to `TRUE`.

However, regardless of the current value for this parameter, 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. For example, the following statement enables writing to a trace file for the session:

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

For the multi-threaded server, because each session using a dispatcher is routed to a shared server process, 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.



For complete information about the ALTER SESSION command, see the *ORACLE7 Server SQL Language Reference Manual*.

Starting the Checkpoint Process

If the Checkpoint process (CKPT) is not present, 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.

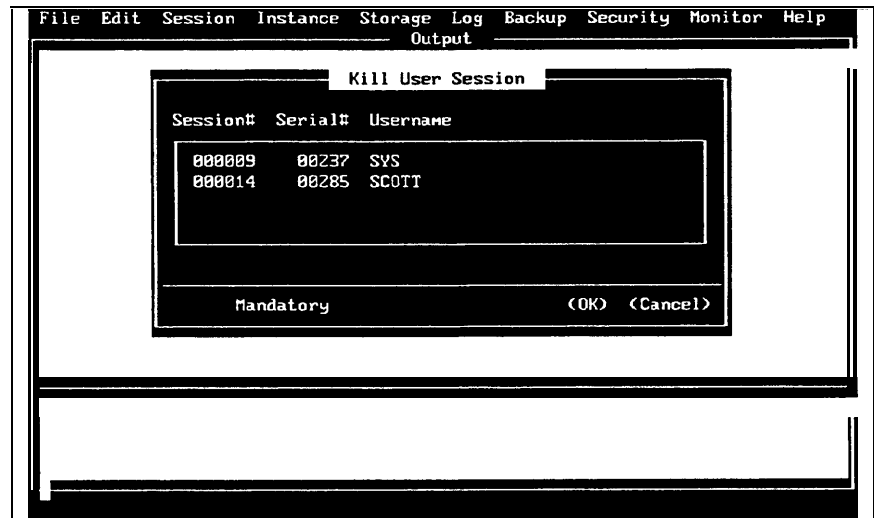
To do so, set the CHECKPOINT_PROCESS parameter in the database's parameter file to TRUE. (Its default is FALSE.)

Killing Sessions

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

Kill a current session using either the Kill User Session dialog box of SQL*DBA or the SQL command ALTER SYSTEM with the KILL SESSION parameter. Figure 4-3 shows the Kill User Session dialog.

FIGURE 4-3
The Kill User Session Dialog



The following statement is the command equivalent of the Kill User Session dialog in Figure 4-3:

```
ALTER SYSTEM KILL SESSION '7, 15';
```

To identify which session to kill, 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. For example, the following query identifies all sessions for the user JWARD:

```
SELECT sid, serial#
   FROM v$session
  WHERE username = 'JWARD';
```

SID	SERIAL#
-----	-----
7	15

When a session is killed, resources (such as locks and memory areas) held by the session are immediately released and available to other sessions.

If the user session is making a SQL call to ORACLE (is "active") when it is killed, the transaction is rolled back and the user immediately receives the following message:

```
ORA-00028: your session has been killed
```

If the session is not making a SQL call to ORACLE (is "inactive") when it is killed, the above message is not returned immediately, but when the user subsequently attempts to use the killed session.

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
```

The view V\$SESSION also shows the STATUS of current sessions. The row for a killed session is removed from V\$SESSION when the user receives the ORA-00028 message.

The following example shows a DBA killing an inactive session:

```
SELECT sid, serial#, status, server
FROM v$session
WHERE username = 'JWARD';
```

SID	SERIAL#	STATUS	SERVER
7	15	INACTIVE	DEDICATED

```
ALTER SYSTEM KILL SESSION '7, 15';
```

```
SELECT sid, serial#, status, server
FROM v$session
WHERE username = 'JWARD';
```

SID	SERIAL#	STATUS	SERVER
7	15	KILLED	PSEUDO

If an active session cannot be interrupted (for example, it is performing network I/O or rolling back a transaction), the session cannot be killed until the operation completes. In this case, the session holds all resources until it is killed. Additionally, the session that issues the ALTER SYSTEM statement to kill a session waits up to 60 seconds for the session to be killed; 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 killed. A session marked to be killed is indicated in V\$SESSION with a status of “KILLED” and a server that is something other than “PSEUDO”.

MANAGING THE ONLINE REDO LOG

This chapter explains how to manage the online redo log. Topics include:

- policies for managing the online redo log
- creating redo log groups and members
- renaming and relocating redo log members
- dropping redo log groups and members
- forcing a redo log switch
- setting checkpoint intervals
- getting information about the online redo log



Parallel

For information about managing the online redo logs of the instances when using the ORACLE Parallel Server, see the *ORACLE7 Parallel Server Administrator's Guide*.

For information about archiving the redo log, see Chapter 17.

Establishing Policies for the Online Redo Log

Consider the following guidelines when configuring a database instance's online redo log.

Mirror the Online Redo Log

The online redo log of a database instance should consist of mirrored groups of online redo log files. Furthermore, members in the same group should be stored on separate disks so that no single disk failure can cause LGWR and the database instance to fail.



Although the ORACLE Server allows mirrored groups to contain different numbers of members, this state should only be temporary, as the result of an abnormal situation such as a disk failure damaging a member of a group; if any group contains only one member, the failure of the disk containing that member could cause ORACLE to halt.

While mirrored groups require extra storage space, the cost of this space is usually insignificant compared to the potential cost of lost data if a disk failure were to destroy a non-mirrored online redo log.

Appropriately Place Online Redo Log Members

With a mirrored online redo log, place members of a group on different disks. This way, if a single disk fails, 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 are archiving the redo log, spread online redo log members across disks to eliminate contention between the LGWR and ARCH 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. This way, there is never contention between LGWR (writing to the members) and ARCH (reading the members).

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

Appropriately Set the Size of Online Redo Log Members

Consider two issues when setting the size of online redo log files: the general size of online redo log files for an instance, and the size of a specific online redo log file compared to other online redo log files of the same database instance.

Online redo log files should be relatively small. If large online redo log files are used and checkpoints only occur at log switches, instance recovery takes a long time because of the large number of redo log entries that must be applied. In most cases, the default size of an online



redo log file is sufficient; the default size of online redo log files is operating system dependent; see your installation or user's guide.

When setting the size of online redo log files, also consider if you will be archiving the redo log. The size of online redo log files should be such 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 per tape and 49% of the tape's storage capacity remains unused. In this case, it would be better to decrease the size of the online redo log files slightly, so that two log groups could be archived per tape.

With mirrored 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.

Choose the Number of Online Log Files

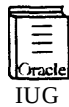
The best way to determine the appropriate number of online redo log files for a database instance is to test different configurations. The optimum configuration has the fewest groups possible without hampering LGWR'S writing redo log information.

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 file. 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.

Three 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 that created the database 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; therefore, 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 parameter of the parameter file can temporarily decrease the maximum number of groups of online redo log files for the duration of the current instance. However, 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 that created the database 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; therefore, 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.



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. See your installation or user's guide for the default and legal values for the MAXLOGFILES and MAXLOGMEMBERS parameters of the CREATE DATABASE statement and the LOG_FILES initialization parameter.

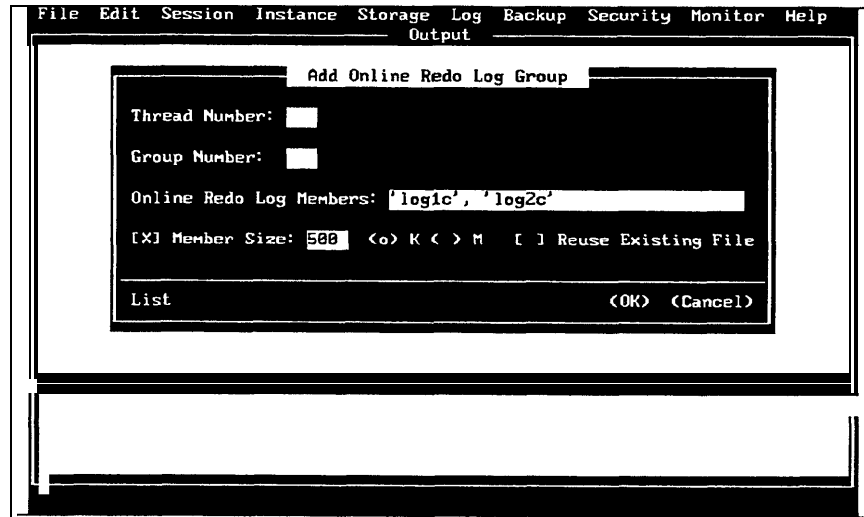
Creating Online Redo Log Groups

You can create groups of online redo log files both as part of database creation and after a database has been created. If you can, plan the online redo log of a database and create all required groups of online redo log files during database creation; see Chapter 2 for information about database creation.

In some cases, you might need to create additional groups 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.

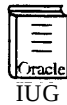
To create a new group of online redo log files, use either the Add Online Redo Log Group dialog of SQL*DBA, or the SQL command ALTER DATABASE with the ADD LOGFILE parameter. Figure 5-1 shows the Add Online Redo Log Group dialog.

FIGURE 5-1
The Add Online Redo Log
Group Dialog



The following statement is the command equivalent of the Add Online Redo Log Group dialog in Figure 5-1:

```
ALTER DATABASE
  ADD LOGFILE ('log1c', 'log2c') SIZE 500K;
```



Note: 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. If you want to reuse an existing operating system file, you do not have to indicate the file size.

In an 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 ('log1c', 'log2c') SIZE 500K;
```

Using group numbers can make administering redo log groups easier. However, the group number must be between 1 and MAXLOGFILES; do not skip redo log file group numbers (that is, do not number your groups 10, 20, 30, and so on), or you will consume unnecessary space in the control files of the database.

Privileges Required to **Create Online Redo** **Log Groups**

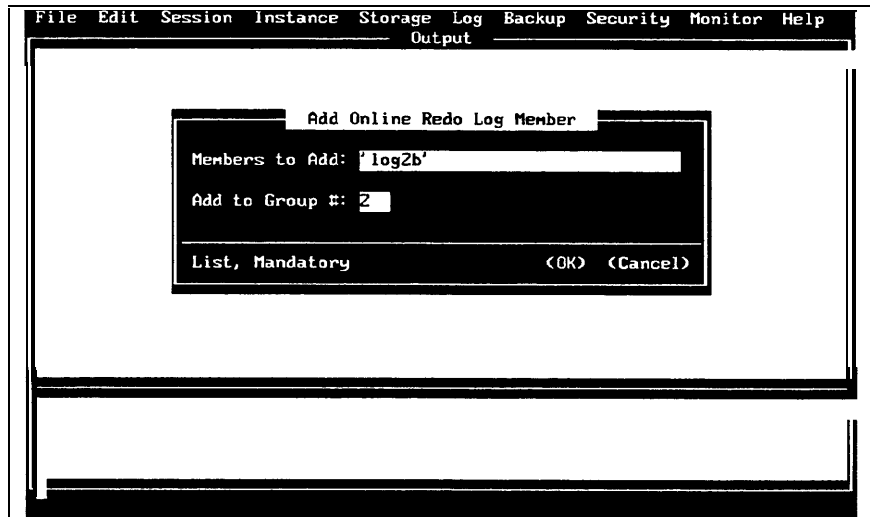
To create new online redo log groups, you must have the ALTER DATABASE system privilege.

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 Add Online Redo Log Member dialog box of SQL*DBA or the SQL command ALTER DATABASE with the ADD LOG MEMBER parameter. Figure 5-2 shows the Add Online Redo Log Member dialog box.

FIGURE 5-2
The Add Online Redo Log
Member Dialog



The following statement is the command equivalent of the Add Online Redo Log Member dialog box in Figure 5-2:

```
ALTER DATABASE  
  ADD LOGFILE MEMBER 'log2b' 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. For example:

```
ALTER DATABASE
  ADD LOGFILE MEMBER 'log2c' TO ('log2a', 'log2b');
```

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

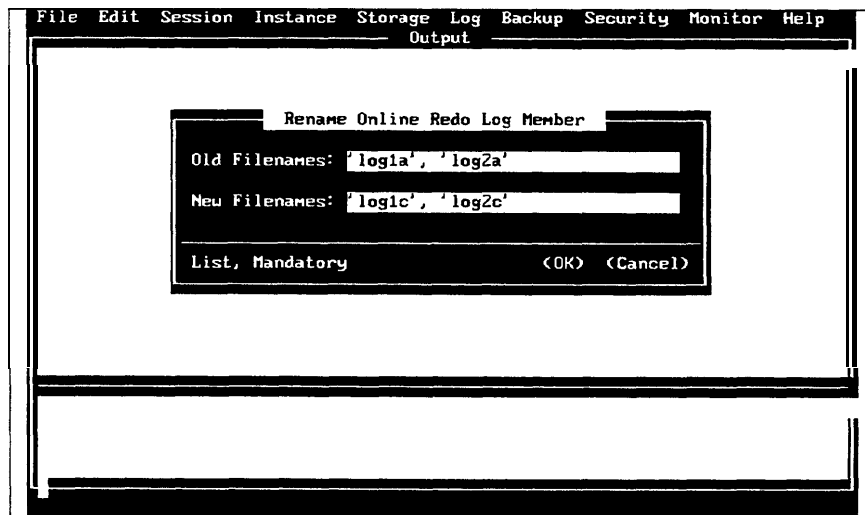
You can rename online redo log members to change their locations. This procedure is necessary, for example, if the disk currently used for some online redo log files is going to be removed, or if data files and a number of online redo log files are stored on the same disk and should be separated to reduce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



Before renaming any online redo log members, ensure that the new online redo log files already exist. The steps given below only modify the internal file pointers in a database's control files; they do not physically rename or create any operating system files. Use the operating system of your computer to copy the existing online redo log files to the new location.

Rename online redo log members with the Rename Online Redo Log Member dialog box of SQL*DBA, or the SQL command ALTER DATABASE with the RENAME FILE parameter. Figure 5-3 shows the Rename Online Redo Log Member dialog box.

FIGURE 5-3
The Rename Online Redo
Log Member Dialog Box



The following statement is the command equivalent of the Rename Online Redo Log Member dialog box in Figure 5-3:

```
ALTER DATABASE
  RENAME FILE 'log1a', 'log2a'
           TO 'log1c', 'log2c';
```

Example Assume the following conditions:

- The database is currently mounted by, but closed to, the instance.
- The online redo log is duplexed: one group consists of the members LOG1A and LOG1B, and the second group consists of the members LOG2A and LOG2B. The files LOG1A and LOG2A are both stored on Disk A, while LOG1B and LOG2B are stored on Disk B.
- The online redo log files located on Disk A must be relocated to Disk C. The new filenames will reflect the new location: LOG1C and LOG2C.

To complete this operation, the following steps are necessary:

1. Backup 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. (See Chapter 18 for database backup procedures.)

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.



Tip: You can execute an operating system command to copy a file without exiting SQL*DBA: either use the Enter System Command dialog box or enter a statement containing the HOST command in the SQL*DBA Input window.

In this example, the files LOG1A and LOG2A on Disk A must be copied to the new files LOG1C and LOG2C on Disk C.

3. Rename the online redo log members.

Use the Rename Online Redo Log Member dialog box or the ALTER DATABASE command with the RENAME FILE clause to rename the online redo log files used by the database.

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.

After renaming or relocating a set of online redo log files, immediately back up the database's control file in case any problems occur from this point forward.

Privileges Required to Rename or Relocate Online Redo Log Members

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 backup the database.

Dropping Online Redo Log Groups

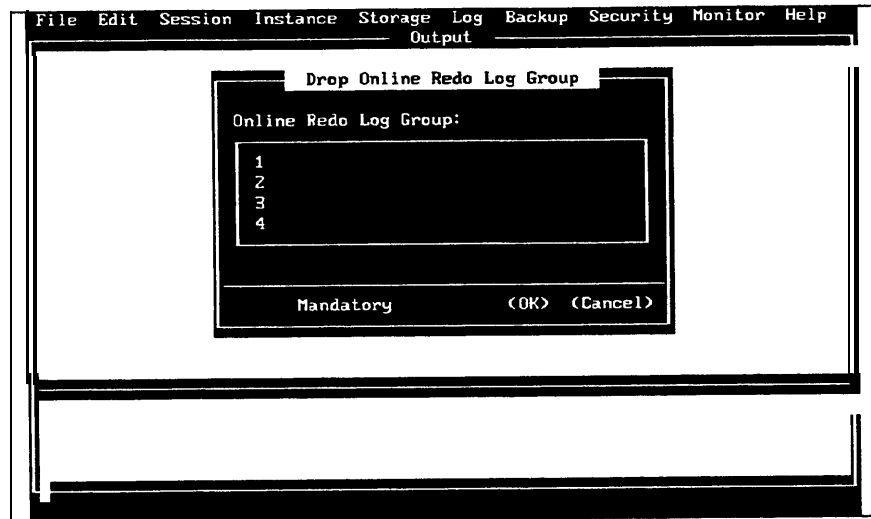
In some cases, you might want to drop an entire group of online redo log members. For example, you might want to reduce the number of groups in an instance's online redo log.

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 A Log Switch" on page 5-14.
- Make sure an online redo log group is archived (if archiving is enabled) before dropping it. To see whether or not this has happened, use SQL*DBA'S List Archive Status menu option (in the Log menu) or the ARCHIVE LOG command with the LIST parameter.

Drop an online redo log group with either the Drop Online Redo Log Group dialog box of SQL*DBA, or the SQL command ALTER DATABASE with the DROP LOGFILE clause. Figure 5-4 shows the Drop Online Redo Log Group dialog box:

FIGURE 5-4
The Drop Online Redo Log
Group Dialog



The following statement is the command equivalent of the Drop Online Redo Log Group dialog box in Figure 5-4:

```
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.

**Privileges Required to
Drop Online Redo Log
Groups**

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

Dropping Online Redo Log Members

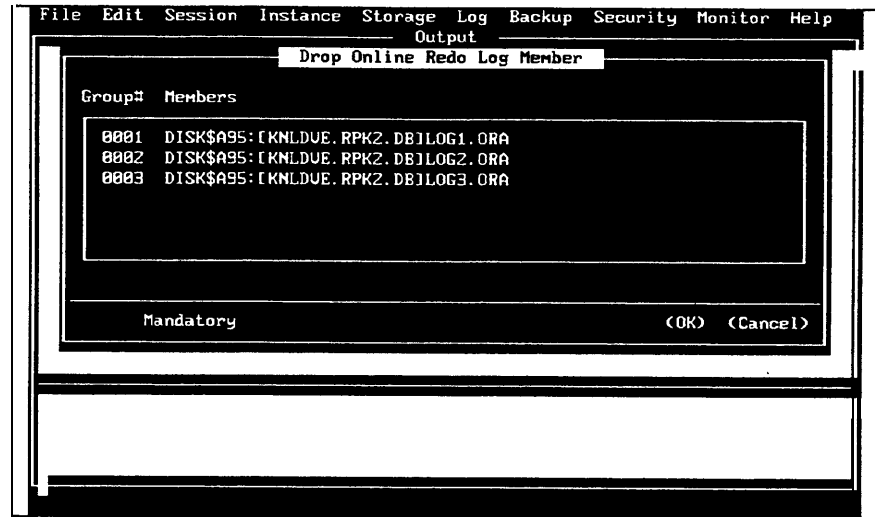
In some cases, you might want to drop one or more specific online redo log members. For example, if a disk failure occurs, you might need to drop all the online redo log files on the failed disk so that ORACLE does not try to write to the inaccessible files. In other situations, particular online redo log files become unnecessary; for example, a file might be stored in an inappropriate location.

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

- It is OK to drop online redo log files so that a mirrored online redo log becomes temporarily unsymmetrical. For example, if you are using 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 table. (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; see "Forcing a Log Switch" on page 5-14.
- 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 SQL*DBA'S List Archive Status menu option or the ARCHIVE LOG command with the LIST parameter.

To drop specific inactive online redo log members, use either the Drop Online Redo Log Member dialog box, or the SQL command ALTER DATABASE command with the DROP LOGFILE MEMBER clause. Figure 5-5 shows the Drop Online Redo Log Member dialog box.

FIGURE 5-5
The Drop Online Redo Log
Member Dialog



The following statement is the command equivalent of the Drop Online Redo Log Member dialog box in Figure 5-5:

```
ALTER DATABASE
  DROP LOGFILE MEMBER 'log3c';
```

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.

Privileges Required to Drop Online Redo Log Members

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

Controlling Checkpoints and Log Switches

A checkpoint is the event during which the Database Writer process (DBWR) writes all modified database buffers in the SGA to the appropriate data files. A log switch is the event during which LGWR stops writing to one online redo log group and starts writing to another. The two events are often connected: an instance takes a checkpoint at each log switch by default. A log switch, by default, takes place automatically when the current online redo log file group fills.

Sometimes, however, you might want checkpoints to be taken more often than only at log switches, or you might want a checkpoint to take place ahead of schedule, without a log switch. You might also want a log switch and checkpoint to take place ahead of schedule, or you might just want a log switch to take place without an accompanying checkpoint. The following sections explain why and how to cause these events.

Setting Database Checkpoint Intervals

When your database uses large online redo log files, you might want additional database checkpoints to take place automatically at predetermined intervals, between the checkpoints that automatically occur at log switches. When more database checkpoints are taken, the time necessary to recover from an instance failure decreases. However, the performance of the ORACLE Server can also decrease as a checkpoint proceeds due to the extra I/O necessary for the checkpoint to complete.

Generally, unless your database consistently requires instance recovery on startup, set database checkpoint intervals so that checkpoints occur only at log switches. If you use small online redo log files, checkpoints already occur at frequent intervals (at each log switch). (For more information about tuning ORACLE with regard to checkpoints, see Chapter 24.)

The frequency of automatic database checkpoints is controlled by the values of the `LOG_CHECKPOINT_INTERVAL` and `LOG_CHECKPOINT_TIMEOUT` parameters.

Setting LOG_CHECKPOINT_INTERVAL

To have database checkpoints only occur at log switches (the default), set the value for the `LOG_CHECKPOINT_INTERVAL` parameter higher than the size of the online redo log files in use. Alternatively, to force additional checkpoints to occur at intervals between two log switches, set the value for the `LOG_CHECKPOINT_INTERVAL` parameter lower than the size of the online redo log files in use.

The value of the LOG_CHECKPOINT_INTERVAL is a number of operating system blocks, not ORACLE data blocks. Therefore, you must know the size, in bytes, of your operating system's blocks. Once you know this, calculate the number of operating system blocks per online redo log file. See your operating system documentation to determine the operating system block size.

Example For example, assume the following conditions:

- All online redo log files of the database instance are 512K.
- The operating system block size is 512 bytes.
- Checkpoints should occur when an online redo log file is half full.

Using this information, you can compute the number of blocks per redo log file as in the following equation:

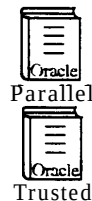
$$\frac{512 \text{ K/redo log file}}{512 \text{ bytes/OS block}} = \text{approximately } 1000 \text{ blocks/redo log file}$$

Now that the approximate number of blocks per online redo log file (1000) is known, the LOG_CHECKPOINT_INTERVAL parameter can be set accordingly in the instance's parameter file.

LOG_CHECKPOINT_INTERVAL=500

Setting LOG_CHECKPOINT_TIMEOUT

To have database checkpoints only occur at log switches (the default), set the value for the LOG_CHECKPOINT_TIMEOUT parameter to zero. Alternatively, to force additional checkpoints to occur at intervals between two log switches, set the value for the LOG_CHECKPOINT_TIMEOUT parameter to a time interval (in seconds) less than the average time it takes to fill an online redo log file. To determine the average time it takes to fill online redo log files, examine the LGWR trace file for messages that indicate the times of log switches.



Note: See the *ORACLE7 Parallel Server Administrator's Guide* for more information about the LOG_CHECKPOINT_TIMEOUT parameter when using the ORACLE Parallel Server. Also, setting this parameter is particularly important when using Trusted ORACLE7 in OS MAC mode; see the *Trusted ORACLE7 Server Administrator's Guide* for more information.

Forcing a Log Switch

You might want to force a log switch to make the currently active group inactive and available for online redo log maintenance operations. For example, you might want to drop the currently active

group, but not be able to do so until the group is inactive. You might also want 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 often useful in configurations with large online redo log files that take a long time to fill.

To force a log switch, use either the Force Log Switch option of the Instance menu of SQL*DBA, or the SQL command ALTER SYSTEM with the SWITCH LOGFILE option. The following statement is the command equivalent of selecting the Force Log Switch option of the Instance menu of SQL*DBA:

```
ALTER SYSTEM SWITCH LOGFILE;
```

Privileges Required to
Force a Log Switch

To force a log switch, you must have the ALTER SYSTEM system privilege.

Forcing a Fast Database Checkpoint Without a Log Switch

In some cases, you might want to force a fast database checkpoint. A fast checkpoint is one which does not involve a log switch; LGWR continues to write to the current online redo log file. A fast checkpoint allows DBWR to write more modified database buffers to disk per I/O on behalf of a checkpoint. Therefore, fewer I/Os (thus less time) are required to complete a fast checkpoint. Force a fast database checkpoint with either the Force Checkpoint option of the Instance menu of SQL*DBA or the SQL command ALTER SYSTEM with the CHECKPOINT option. The following statement is the command equivalent of the Force Checkpoint menu option:

```
ALTER SYSTEM CHECKPOINT;
```

Omitting the GLOBAL option allows you to force a checkpoint for only the connected instance, while including it forces a checkpoint for all instances of the database. Forcing a checkpoint for only the local instance is useful only with the ORACLE Parallel Server; see the *ORACLE7 Parallel Server Administrator's Guide* for more information. In a non-Parallel Server configuration, global and local checkpoints are identical.



Parallel

Privileges Required to
Force a Checkpoint

To force a database checkpoint, you must have the ALTER SYSTEM system privilege.

Listing Information about the Online Redo Log

Use the V\$LOG, V\$LOGFILE, 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.

For example, 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 like that below:

```
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, the file is in use.

6

MANAGING CONTROL FILES

This chapter explains how to create and maintain the control files for your database. Topics include:

- guidelines for managing control files
- initially creating control files
- adding, renaming, and relocating control files
- dropping control files

Guidelines for Control Files

Consider the following guidelines when managing the control files for a database.

Name Control Files

The `CONTROL_FILES` parameter in the parameter file of a database indicates one or more names for the control files. The instance startup procedure recognizes and opens all the listed files; the instance maintains all listed control files during database operation.

All files listed for the `CONTROL_FILES` parameter are written to, as necessary, by the ORACLE Server during database operation.

Mirror 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. By using mirrored control files, you avoid unnecessary problems if a disk failure occurs on the database server.

The following list describes the behavior of mirrored 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.

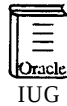
The only disadvantage of having multiple control files is that all operations that update the control files (such as adding a data file or checkpointing the database) can take slightly longer. However, this difference is usually insignificant (especially for operating systems that can perform multiple,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

Oracle Corporation 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 mirrored. 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 Control File Size



Typical control files are small. The main determinants of a control file's size are the values set for the MAXDATAFILES, MAXLOGFILES, MAXLOGMEMBERS, MAXLOGHISTORY, and MAXINSTANCES parameters of 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. The maximum control file size is operating system specific; see your installation or user's guide.

Creating Control Files

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, multiple copies are created. However, 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. The ways to create control files are explained below.

Creating Initial Control Files



You create the initial control files of an ORACLE database by specifying one or more control file names 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 command, 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 release of new version 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 no files are specified for CONTROL_FILES before database creation, a

default filename is used; the default name is also operating system specific.

You can later 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 the next section for details.

Creating Additional Copies of the Control File, 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 mirror additional copies of the current control file for a database, or to move a control file, use the following steps:

1. Shut down the database.
2. Exit SQL*DBA.
3. Copy an existing control file to a different location, using operating system commands.
4. Edit the CONTROLFILES parameter in the database's parameter file to add the new control file's name, or to change the existing control file name.
5. Restart SQL*DBA.
6. Restart the database.

Creating a New Control File

You can create a new control file for a database using the CREATE CONTROLFILE command. This is useful 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 database. As another example, you might need to change one of the previously mentioned parameters if the original setting is too low.)

You create a new control file using the CREATE CONTROLFILE command. Use the following procedure to create a new control file.



Note: The CREATE CONTROLFILE command can potentially damage specified data files and online redo log files; omitting a filename can cause you to lose the data in that file, or even to lose access to the entire database. Use caution when using this command and be sure to use the steps in the following procedure:

1. Make a list of all data files and online redo log files of the database (see “Listing Database Files Before Backup” on page 18-6). If you followed the recommendations for database backups, you should already have a list of data files 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, then do your best to locate all of the data files 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 makeup the SYSTEM tablespace, you might not be able to recover the database.)
2. If the database is open, shutdown the database with normal priority, if possible. Use the immediate or abort options only as a last resort.
3. Backup all data files and online redo log files of the database using the procedure described in “Performing Full Backups” on page 18-7.
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 command. For example, 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;
```

In 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 in step 8.

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 to indicate all of the control files created in Steps 5 and 6 (not including the backup control file) in the CONTROL_FILES parameter.
8. 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 (in step 5), you can recover the database with complete, closed database recovery (see page 19-18). However, if the new control file was created using the RESETLOGS option, you must specify USING BACKUP CONTROL FILE (see page 19-23); if you have lost online or archived redo logs or data files, use the procedures for recovering those files (see “Loss of Online Redo Log Files” on page 19-29 and “Loss of Data Files” on page 19-28).
9. 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**, use the Open Database option of the Instance menu of SQL*DBA.
 - **If you specified RESETLOGS when creating the control file**, use the ALTER DATABASE command, indicating RESETLOGS.

The database is now open and available for use.

Checking for Missing or Extra Files

After creating a new control file and using it to open the database, check the ALERT file to see if ORACLE has detected inconsistencies between the data dictionary and the control file, such as data files that the control file lists but the data dictionary does not include.

If a data file listed in the data dictionary is not mentioned in the new control file, ORACLE adds the file to the control file and renames it to MISSINGnnnn in the data dictionary; you should rename this file to a new, appropriate name. In contrast, if a data file 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 file names.



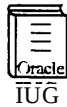
Note: See the *ORACLE7 Server SQL Language Reference Manual* for additional information about the CREATE CONTROLFILE command.

Dropping Control Files

You can drop control files from the database; for example, you might want to do this if the location of a control file is inappropriate. However, the database must always have at least two control files at all times.

To drop a control file from database use, do as follows:

1. Shut down the database.
2. Exit SQL*DBA.
3. Edit the CONTROLFILES parameter in the database's parameter file to delete the old control file's name.
4. Restart SQL*DBA.
5. Restart the database.



The operation above does not physically delete the unwanted control file from the disk; use operating system commands to delete the unnecessary file once you have completed the steps above. (See your installation or user's guide.)

PART

III

DATABASE STORAGE

MANAGING THE SIZE AND FILES OF THE DATABASE

This chapter explains how to control the amount of space in your database through managing tablespaces and data files. Topics include:

- policies for managing tablespaces and data files
- creating tablespaces and data files
- setting storage parameters for tablespaces
- altering tablespace and data file availability
- renaming and relocating data files
- dropping tablespaces and data files
- viewing information on tablespaces and data files

Data files can also be created as part of database recovery from a media failure. Refer to Chapter 19 for information.



Trusted

If you are using Trusted ORACLE in DBMS MAC mode or OS MAC mode, see the *Trusted ORACLE7 Server Administrator's Guide* for additional information on tablespaces and data files.

Policies for Managing Tablespaces and Data Files

Before working with the tablespaces and data files of an ORACLE database, consider the information in the following sections.

Use 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' data files on separate disk drives to reduce I/O contention.
- Separate rollback segment data from user data, preventing a single disk failure from causing the 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, leaving the rest available to users.

However, some operating systems set a limit on the number of files that can be simultaneously open; these limits indirectly affect the number of tablespaces that can be simultaneously online. To avoid exceeding this 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 data files rather than many small data files.

Review your data in light of these advantages and decide how many tablespaces you will need for your database design.

Number of Data Files

At least one data file is required for the SYSTEM tablespace of a database; a small system might have just a single data file. In general, few large files are better than many small files, because large files mean that fewer files need to be open at the same time.



IUG

You can add data files to tablespaces later, subject to several operating system specific data file limits (see your installation or user's guide):

operating system limit	Each operating system sets a limit on the maximum number of files per process. Regardless of all other limits, more data files cannot be created if the operating system limit of open files has been reached.
ORACLE system limit	ORACLE imposes a maximum limit on the number of data files for any ORACLE database opened by any instance.
control file upper bound	At database creation, you must indicate the maximum number of data files expected for the database so that an adequate amount of space can be reserved in the database's control file. You set this limit with the MAXDATAFILES parameter in the CREATE DATABASE statement. This maximum cannot exceed the ORACLE system limit or any operating system limit. If you are not sure how to set this parameter, use a high number to avoid unnecessary limitation. The default value is operating system specific.
instance or SGA upper bound	When starting an ORACLE instance, the database's parameter file indicates the amount of SGA space to reserve for data file information; the maximum number of data files is controlled by the DB_FILES parameter. This limit applies only for the life of the instance. DB_FILES can temporarily reduce the control file upper bound, but cannot raise it. The default value is the value of the control file upper bound.



Note: With the ORACLE Parallel Server, all instances must set the instance data file upper bound to the same value. Refer to the *ORACLE7 Parallel Server Administrator's Guide* for more information.

The use of DB_FILES and MAXDATAFILES is optional. If neither is used, the default maximum number of data files is the operating system dependent ORACLE system limit.

Set the Size of Data Files

The first data file (in the original SYSTEM tablespace) must be at least 2M, 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); refer to the installation instructions for other products you plan to install.

**Place Data Files for
Maximum Performance**

Tablespace location is determined by the physical location of the data files 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 Data Files
Separately From Redo
Log Files**

Data files should not be stored on the same disk drive that stores the database's redo log files. If the data files and redo log files are stored on the same disk drive and that disk drive fails, the files can be used in your database recovery procedures.

If you mirror your redo log files, then the likelihood of your losing all of your redo log files is low, so you can store data files on the same drive as some redo log file.

**Specify 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, but 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 can specify different storage parameters for an unusual or exceptional object when creating that object.

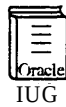
To follow this recommendation, you need to estimate the size of the typical object of the tablespace. See Chapter 8 for formulas for estimating the sizes of objects.

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.

Assign Tablespace Quotas to Users

Grant to users who will be creating tables, clusters, snapshots, indexes, and so on, the privileges (abilities or rights) 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 is also responsible for assigning tablespace quotas, as necessary, to database users. To learn more about assigning tablespace quotas to database users, refer to Chapter 11.

Initially Creating Tablespaces and Data Files



The steps for creating data files vary by operating system. On most operating systems, you indicate the size and fully specified filenames for data files when creating a database, creating a new tablespace, or altering a tablespace by adding additional data files; in each situation, ORACLE automatically allocates and formats the data files as specified. However, on some operating systems, you must create the data files before installation. Check your installation or user's guide for the steps appropriate for your system.

The first tablespace in any database is always the SYSTEM tablespace. Therefore, the first data files of any database are automatically allocated for the SYSTEM tablespace during database creation.

Creating Additional Tablespaces

You can create a new tablespace for any of the following reasons:

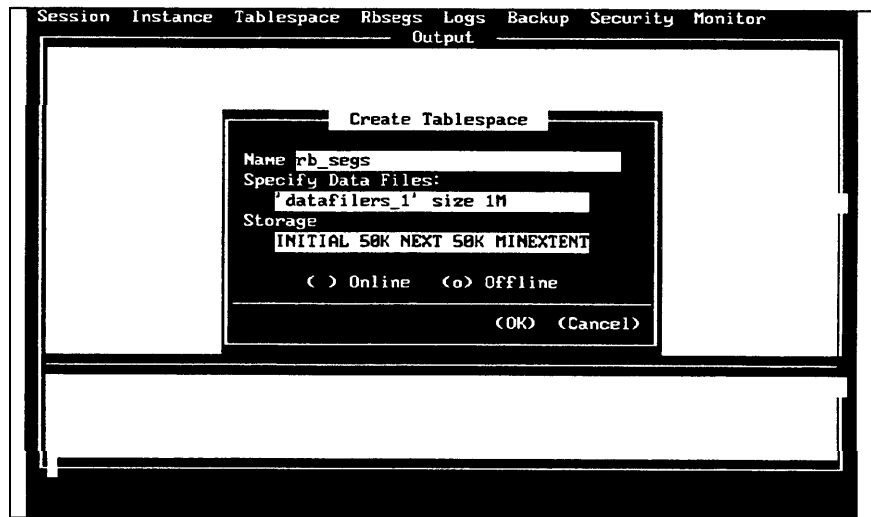
- You want to allocate more disk storage space for the associated database, enlarging the database.
- You need to create a logical storage structure in which to store a specific type of data separate from other database data.

Note that to increase the total size of the database you can alternatively add a data file to an existing tablespace, rather than adding a new tablespace. See “Adding Data Files to a Tablespace” on page 7-7.

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). Query the data dictionary to determine how many rollback segments currently exist in a database; see page 9-16 for more information.

To create a new tablespace, use either the Create TableSpace dialog box of SQL*DBA or the SQL command CREATE TABLESPACE. Figure 7-1 shows the Create Tablespace dialog box.

FIGURE 7-1
The Create Tablespace Dialog



This example creates a new tablespace named RB_SEGS (to hold rollback segments for the database) with the following options:

- The data of the new tablespace is to be contained in a single data file, 1M 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 is the command equivalent of the Create Tablespace dialog box in Figure 7-1:

```
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 data files are created in the current directory of the database server.

Privileges Required to Create Tablespaces

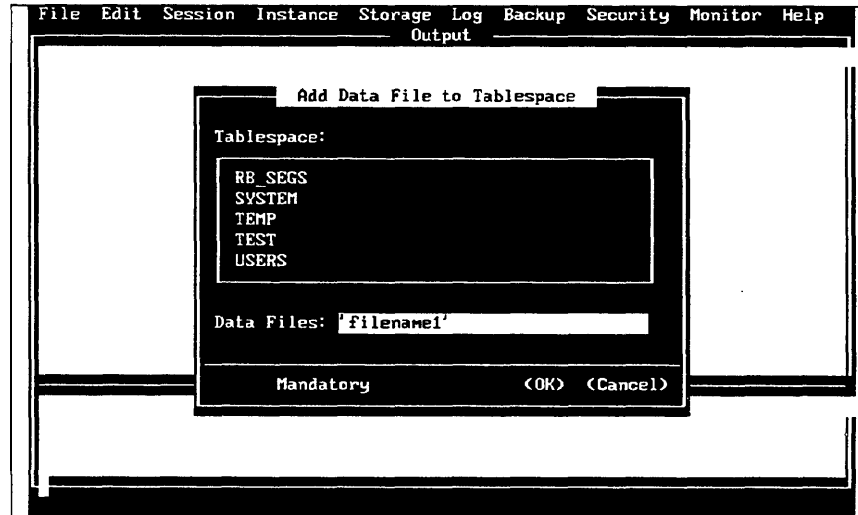
To create a tablespace, you must have the CREATE TABLESPACE system privilege.

Adding Data Files to a Tablespace

You can create and add data files to a tablespace to increase the total amount of disk space allocated for the tablespace, and consequently the database.

Add data files to a tablespace with either SQL*DBA'S Add Data Files to Tablespace dialog or the SQL command ALTER TABLESPACE. Figure 7-2 shows the Add Data File to Tablespace dialog box of SQL*DBA.

FIGURE 7-2
The Add Data File to
Tablespace Dialog



The example in Figure 7-2 creates a new data file for the RB_SEGS tablespace.

The following statement is the command equivalent of the Add Data File to Tablespace dialog box in Figure 7-2:

```
ALTER TABLESPACE rb_segs  
  ADD DATAFILE 'filename1' SIZE 1M;
```

If you add new data files to a tablespace and do not fully specify the filenames, ORACLE creates the data files 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 will be overwritten.

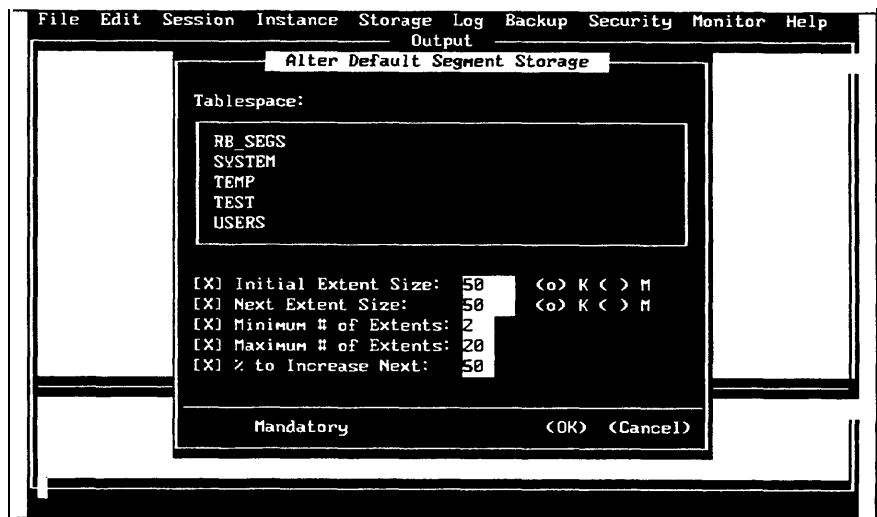
Privileges Required to Add Data Files

To add data files to a tablespace, you must have the ALTER TABLESPACE system privilege.

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 do so, use the Alter Default Segment Storage dialog box of SQL*DBA. Figure 7-3 shows the Alter Default Segment Storage dialog box.

FIGURE 7-3
The Alter Default Segment Storage Dialog



The following statement is the command equivalent of the Alter Default Segment Storage dialog box in Figure 7-3:

```
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.

Privileges Required to Alter Tablespace Storage Settings

To alter the default storage parameters of a tablespace, you must have the ALTER TABLESPACE system privilege.

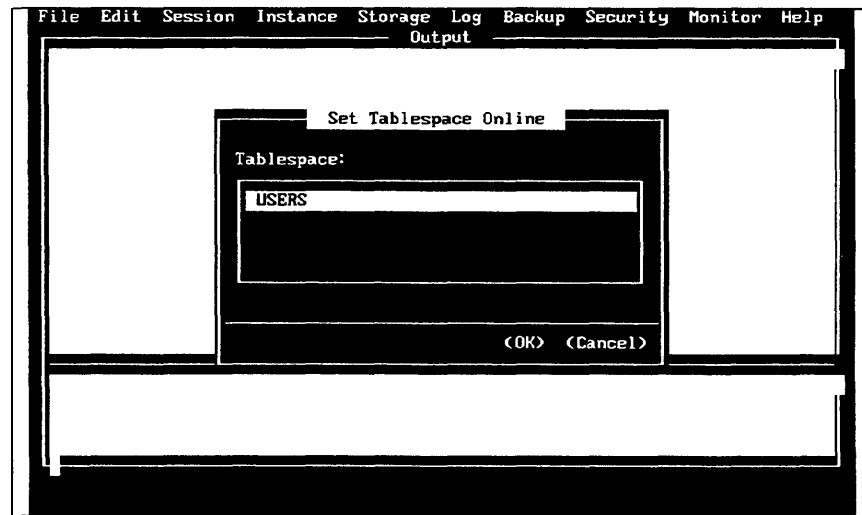
Altering Tablespace Availability

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. The following sections explain how to do this.

Bringing Tablespaces Online

Bring an offline tablespace online while the database is open using either the Set Tablespace Online dialog box of SQL*DBA or the SQL command ALTER TABLESPACE. Figure 7-4 shows the Set Tablespace Online dialog box.

FIGURE 7-4
The Set Tablespace Online Dialog



The Set Tablespace Online dialog box in Figure 7-4 brings the USERS tablespace online.

The following statement is the command equivalent of the Set Tablespace Online dialog box in Figure 7-4:

```
ALTER TABLESPACE users 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 command), you must first perform media recovery on the tablespace before bringing it online. Otherwise, ORACLE returns an error and the tablespace remains offline.

Privileges Required to
Bring Tablespaces Online

To bring a tablespace online, you must have the MANAGE
TABLESPACE system privilege.

Taking Tablespaces Offline

You can take a tablespace offline with any of the following three priorities:

normal offline	A tablespace can be taken offline <i>normally</i> if no error conditions exist for any of the data files of the tablespace; no data file in the tablespace can be currently offline as the result of a write error. In this case, ORACLE takes a checkpoint for all data files of the tablespace as it takes them offline.
temporary offline	A tablespace can be taken offline <i>temporarily</i>, even if error conditions are indicated for one or more files of the tablespace. In this case, ORACLE takes offline the data files that are not already offline, checkpointing them as it does so. If no files are offline but you use the temporary option anyway, 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 <i>immediately</i> , without ORACLE’s taking a checkpoint on any of the data files. In this case, 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.



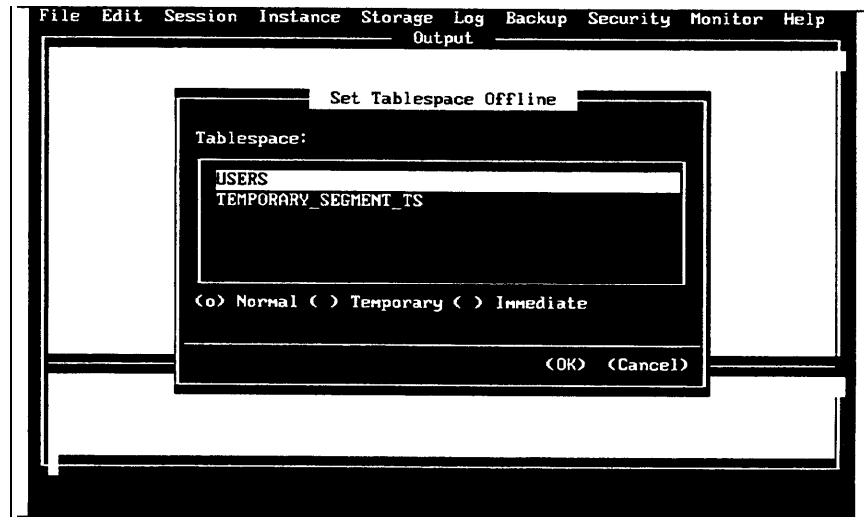
If you must take a tablespace offline, use the normal option (the default) if possible; this guarantees that the tablespace will not require recovery to comeback online, even if you reset the redo log sequence (using an ALTER DATABASE OPEN RESETLOGS statement after incomplete media recovery) before bringing 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.

Note: Before taking an online tablespace offline, verify that the tablespace contains no active rollback segments. See “Taking Rollback Segments Offline” on page 9-12.

Take an online tablespace offline while the database is open with either the Set Tablespace Offline dialog box of SQL*DBA or the SQL command ALTER TABLESPACE. Figure 7-5 shows the Set Tablespace Offline dialog box.

FIGURE 7-5
The Set Tablespace
Offline Dialog



The example in Figure 7-5 takes the USERS tablespace offline normally. The Normal, Temporary, and Immediate radio buttons determine the priority with which to take the specified tablespace offline.

The following statement is the command equivalent of the Set Tablespace Offline dialog box in Figure 7-5:

```
ALTER TABLESPACE users OFFLINE NORMAL;
```

Privileges Required to
Take Tablespaces Offline

To take a tablespace offline, you must have the MANAGE
TABLESPACE system privilege.

Bringing Data Files Online and Offline

In very rare situations, you might need to bring specific data files online (make them available) or take specific files offline (make them unavailable). For example, when ORACLE has problems writing to a data file, it can automatically take the data file offline. You might need to take the damaged data file offline or bring it online manually.

Note: You can make all data files 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.

Bringing Data Files Online or Offline in ARCHIVELOG Mode

Bring an individual data file online or offline with the SQL command ALTER DATABASE, using the DATAFILE parameter.

Note: To use this option of the ALTER DATABASE command, the database must be in ARCHIVELOG mode. This requirement prevents you from accidentally losing the data file, since taking the data file offline while in NOARCHIVELOG mode is likely to result in losing the file. If the database is in NOARCHIVELOG mode and you do want to drop the file, see “Taking Data Files Offline in NOARCHIVELOG Mode,” below.

For example, the following statement brings a data file online:

```
ALTER DATABASE DATAFILE 'filename' ONLINE;
```

For more information about bringing data files online during media recovery, refer to Chapter 19.

Taking Data Files Offline in NOARCHIVELOG Mode

To take a data file 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 data file offline and drop it immediately. It is useful, for example, if the data file contains only data from temporary segments and has not been backed up, and the database is in NOARCHIVELOG mode.

For example, the following statement brings the specified data file offline:

```
ALTER DATABASE DATAFILE 'filename' OFFLINE DROP;
```

Privileges Required to Take a Data File On- or Offline

To bring a data file 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.

Renaming and Relocating Data Files

You can rename data files to change either their names or locations. ORACLE provides options to make the following changes:

- Rename and relocate data files in a single offline tablespace (for example, FILENAME1 and FILENAME2 in TBSPACE1) while the rest of the database is open.
- Rename and relocate data files in several tablespaces simultaneously (for example, FILE1 in TBSP1 and FILE2 in TBSP2) while the database is mounted but closed.

The following sections explain each option in detail.

Note: To rename or relocate data files of the SYSTEM tablespace, you must use the second option, because you cannot take the SYSTEM tablespace offline.

Renaming and relocating data files with these procedures only change the pointers to the data files, 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 data files involve several steps. Read the steps and examples carefully before performing these procedures.

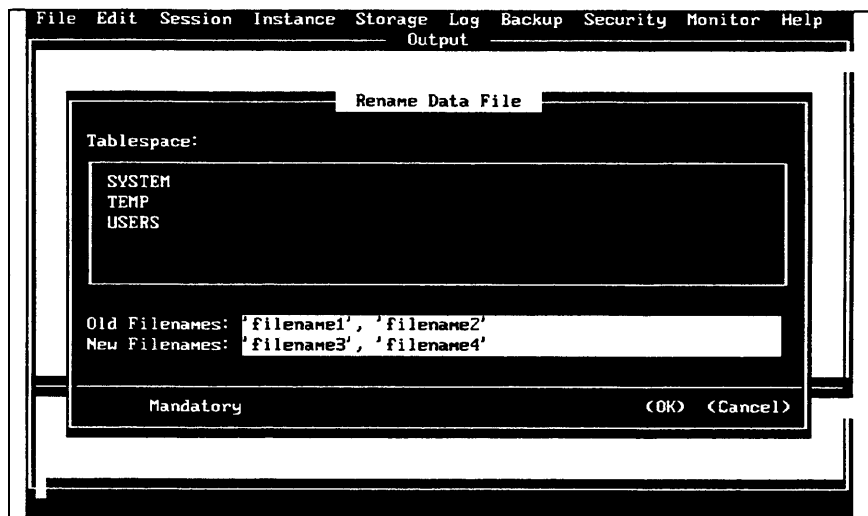
Renaming and Relocating Data Files for a Single Tablespace

Rename or relocate data files from a single tablespace with the following steps:

1. Take the non-SYSTEM tablespace that contains the data files offline
2. Copy the data files to the new location or new names using the operating system.
3. Make sure that the new, fully specified filenames are different from the old filenames.
4. Use either the Rename Data File dialog of SQL*DBA or the SQL command ALTER TABLESPACE with the RENAME DATAFILE option to change the filenames within the database.

Figure 7-6 shows the Rename Data File dialog box.

FIGURE 7-6
The Rename Data File Dialog



The example in Figure 7-6 renames the data files FILENAME1 and FILENAME2 to FILENAME3 and FILENAME4, respectively.

The following statement is the command equivalent of the Rename Data File dialog box in Figure 7-6:

```
ALTER TABLESPACE users
  RENAME DATAFILE 'filename1', 'filename2'
                TO 'filename3', 'filename4';
```

With either option, 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 data files. In particular, specify the old filename exactly as it appears in the DBA_DATA_FILE view of the data dictionary.

Privileges Required to
Rename Data Files of a
Single Tablespace

To rename data files of a single tablespace, you must have the ALTER TABLESPACE system privilege.

Renaming and Relocating Data Files for Multiple Tablespaces

You can rename and relocate data files of one or more tablespaces with the SQL command ALTER DATABASE with the RENAME FILE option. This option is the only choice if you want to rename or relocate data files of several tablespaces in one operation, or rename or relocate data files of the SYSTEM tablespace. If the database must remain open, consider instead the procedure outlined in the previous section.

Rename or relocate data files for multiple tablespaces with the following steps:

1. Ensure that the database is mounted but closed.
2. Copy the data files to be renamed to their new locations and new names, using the operating system.
3. Make sure the new copies of the data files have different fully specified filenames from the data files currently in use.
4. Use the SQL command ALTER DATABASE to rename the file pointers in the database's control file. For example, the following statement renames the data files FILENAME1 and FILENAME2 to FILENAME3 and FILENAME4, respectively:

```
ALTER DATABASE
  RENAME FILE 'filename1', 'filename2'
  TO 'filename3', 'filename4';
```

Note: Always provide complete filenames (with paths), so the RENAME FILE option properly identifies the old and new data files.

Privileges Required to
Rename Data Files of
Multiple Tablespaces

To rename data files of several tablespaces in one operation or to rename data files of the SYSTEM tablespace, you must have the ALTER DATABASE system privilege.

Example of Relocating a Data File

Assume the following conditions:

- An open database has a tablespace named USERS that is comprised of data files located on the same disk of a computer.
- The data files of the USERS tablespace are to be relocated to a different disk drive.
- You are currently connected to the open database as INTERNAL while using SQL*DBA. (See “Connecting as INTERNAL” on page 1-4.)

To complete this operation, the following steps are necessary:

1. Identify the data filenames of interest.

If you are not sure of the filenames for the data files that constitute a tablespace, query the data dictionary view DBA_DATA_FILES. For example, the following query lists the data filenames 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. Refer to Appendix B for more information about the DBA_DATA_FILES data dictionary view.

2. Backup the database.

Before making any structural changes to a database, such as renaming and relocating the data files of one or more tablespaces, always completely back up the database. (See Chapter 18 for database backup procedures.)

3. Take the tablespace offline, or shut down the database and restart and mount it, leaving it closed.

Either option closes the data files that constitute the USERS tablespace. (See “Taking Tablespaces Offline” on page 7-10 for information about taking a tablespace offline, and Chapter 3 for information about mounting a database without opening it.)

4. Copy the data files.

Once the USERS tablespace is offline, or the database is mounted and closed, use operating system commands to make copies of the current data files at the new location. For this example, the existing files FILENAME1 and FILENAME2 are copied to FILENAME3 and FILENAME4. (See your operating system documentation for information on how to copy files.)



Tip: You can execute an operating system command to copy a file without exiting SQL*DBA by using the Enter System Command dialog box or the HOST command in the SQL*DBA input window.

5. Rename the data files within ORACLE.

The data file pointers for the files that comprise 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 USERS tablespace is offline but the database is open, use the Rename Data Files of Tablespace dialog box or ALTER TABLESPACE ... RENAME DATAFILE command.
- If the database is mounted but closed, use the ALTER DATABASE ... RENAME FILE command.

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. (See “Bringing Tablespaces Online” on page 7-9.)

If the database is mounted but closed, open the database.

7. Backup the database.

After making any structural changes to a database, always backup the database immediately and completely.

Dropping Tablespaces and Data Files

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.



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.

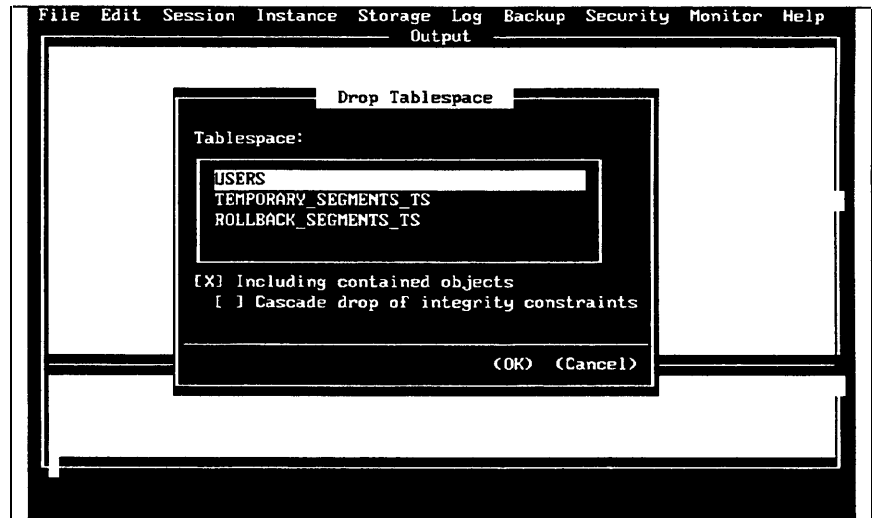
When you drop a tablespace, only the file pointers in the control files of the associated database are dropped; the data files that constituted the dropped tablespace continue to exist. To free previously used disk space, delete the data files of the dropped tablespace using the appropriate commands 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. (See “Taking Tablespaces Offline” on page 7-10.)

After a tablespace is dropped, the tablespace's entry remains in the data dictionary (in the DBA_TABLESPACES view), but the tablespace's status is changed to INVALID.

Drop a tablespace using either the Drop Tablespace dialog box of SQL*DBA or the SQL command DROP TABLESPACE. Figure 7-7 shows the Drop Tablespace dialog box.

FIGURE 7-7
The Drop Tablespace Dialog



The example in Figure 7-7 drops the USERS tablespace, including the segments in the tablespace.

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.

The following statement is the command equivalent of the Drop Tablespace dialog box in Figure 7-7:

```
DROP TABLESPACE users INCLUDING CONTENTS;
```

Use the CASCADE CONSTRAINTS option to cascade the drop of the FOREIGN KEY constraints in the child tables.

Privileges Required to Drop Tablespaces

To drop a tablespace, you must have the DROP TABLESPACE system privilege.

Viewing Information About Tablespaces and Data Files

The following data dictionary views provide useful information about the tablespaces and data files of a database:

- USER_EXTENTS, DBA_EXTENTS (See Chapter 8.)
- USER_SEGMENTS, DBA_SEGMENTS (See Chapter 8.)
- USER_FREE_SPACE, DBA_FREE_SPACE
- DBA_USERS (See Chapter 11.)
- DBA_TS_QUOTAS (See Chapter 11.)
- 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 Guide. 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.

Example 1 Listing Tablespaces and Default Storage Parameters

To list the names and default storage parameters of all tablespaces in a database, use the following query on the DBA_TABLESPACES view:

```
SELECT tablespace_name "TABLESPACE",
       initial_extent  "INITIAL_EXT",
       next_extent    "NEXT_EXT",
       min_extents    "MIN_EXT",
       max_extents    "MAX_EXT",
       pct_increase
FROM   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

Example 2 Listing the Data Files and Associated Tablespaces of a Database

To list the names, sizes, and associated tablespaces of a database, enter the following query on the DBA_DATA_FILES view:

```
SELECT file_name, bytes, tablespace_name
FROM   sys.dba_data_files;
```

FILE_NAME	BYTES	TABLESPACE_NAME
-----	-----	-----
filename1	10240000	SYSTEM
filename2	10240000	USERS
filename3	20480000	USERS

Example 3
Listing the Free Space
(Extents) of Each
Tablespace

To see the amount of space available in the free extents of each tablespace in the database, enter the following query:

```
SELECT tablespace_name, 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	SUM
SYSTEM	1	2	2928	115	1521.5	3043

SUM shows the amount of free space in each tablespace, PIECES shows the amount of fragmentation in the data files 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.

Example 4
Listing Status
Information About
Data Files

The view V\$DATAFILE contains status information about data files of a database. For example:

```
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 data file; the first data file in the SYSTEM tablespace, created with the database, is always file 1. STATUS lists other information about a data file. If a data file is part of the SYSTEM tablespace, its status is SYSTEM (unless it requires recovery). If a data file in a non-SYSTEM tablespace is online, its status is ONLINE. If a data file in a non-SYSTEM tablespace is offline, its status can be either OFFLINE or RECOVER. CHECKPOINT lists the final SCN written for a data file's most recent checkpoint.

8

MANAGING SCHEMA
OBJECTS

This chapter explains how to manage schema objects: tables, indexes, views, clusters, hash clusters, synonyms, sequences, stored procedures, packages, and database triggers. The following topics are included:

- estimating the size of, setting storage parameters for, and creating schema objects
- altering and dropping schema objects



Trusted

Note: If you are using Trusted ORACLE, see the *Trusted ORACLE7 Server Administrator's Guide* for additional information on managing schema objects in that environment.

Guidelines for Managing Schema Objects

This section provides guidelines for the following aspects of schema object management:

- managing how the space of data blocks is used
- setting storage parameters of object segments
- understanding the space requirements of the various datatypes

Managing the Space Usage of Data Blocks

The following sections explain how to 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

Specifying PCTFREE

The default for PCTFREE is 10 percent; you may use any integer between 0 and 99, inclusive, as long as the sum of PCTFREE and PCTUSED does not exceed 100. (If you set PCTFREE to 99, ORACLE will put at least one row in each block, regardless of the row's size. If rows are very small and blocks very large, even more than one row may fit.)

A lower PCTFREE has the following effects:

- reserves less room for updates to 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)
- increases processing costs because ORACLE must frequently reorganize blocks as their free space area becomes filled with new or updated data
- potentially increases processing costs and space required if updates to rows or index entries cause rows to grow and span blocks

A low PCTFREE might be suitable, for example, for a segment that is rarely changed.

A higher 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)

- lessens processing costs because blocks infrequently need reorganization of their free space area
- may improve update performance, because ORACLE does not need to chain row pieces as frequently, if ever

A high PCTFREE is suitable, for example, for segments that are frequently INSERTed.

In setting PCTFREE, understand the nature of the table or index data. 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 longer, 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 tradeoff between densely packed data and good update performance.

PCTFREE also affects the performance of queries on tables with uncommitted transactions belonging to other users. Assuring read consistency may cause frequent reorganization of the free space area in a block if the free area is small.

PCTFREE for Non-Clustered Tables If the data in the rows of a non-clustered 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 non-clustered 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 Indexes infrequently require the use of free space for updates to index data. Therefore, you can usually set PCTFREE for index segments very low (for example, 5 or lower).

Specifying PCTUSED

Once 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. ORACLE tries to keep a data block at least PCTUSED full. The percent is of block space available for data after overhead is subtracted from total space.

The default for PCTUSED is 40 percent; you may specify any integer between 0 and 99, inclusive, as long as the sum of PCTUSED and PCTFREE does not exceed 100.

A lower PCTUSED has the following effects:

- on the average, keeps blocks less full than a higher PCTUSED

- reduces processing costs incurred during UPDATE and DELETE statements for moving a block to the free list when it has fallen below that percentage of usage
- increases the unused space in a database

A higher PCTUSED has the following effects:

- on the average keeps blocks fuller than a lower PCTUSED
- improves space efficiency
- increases processing cost during INSERTS and UPDATES

Choosing 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 is less than 100, then the ideal compromise of space utilization and I/O performance is a sum of PCTFREE and PCTUSED that differs from 100 by the percentage of space in the available block that an average row occupies. For example, assume that the data block size is 2048 bytes, minus 100 bytes of overhead, leaving 1948 bytes available for data. If an average row requires 195 bytes, or 10% of 1948, then an appropriate combination of PCTUSED and PCTFREE that sums to 90% would make the best use of database space.
- If the sum equals 100, then ORACLE attempts to keep no more than PCTFREE free space, and the processing costs are highest.
- Fixed 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 illustrate specifying values for PCTFREE and PCTUSED in given scenarios.

Example 1

Scenario: Common activity includes UPDATE statements that increase the size of the rows.

Settings: PCTFREE = 20
PCTUSED = 40

Explanation: PCTFREE is set to 20 to allow enough room for rows that increase in size as a result of updates. PCTUSED is set to 40 so that less processing is

done during high update activity, thus improving performance.

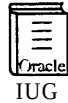
Example 2	Scenario:	Most activity includes INSERT and DELETE statements, and UPDATE statements that do not increase the size of affected rows.
	Settings:	PCTFREE = 5 PCTUSED = 60
	Explanation:	PCTFREE is set to 5 because most UPDATE statements do not increase row sizes. PCTUSED is set to 60 so that space 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 = 90
	Explanation:	PCTFREE is set to 5 because UPDATE statements are rarely issued. PCTUSED is set to 90 so that more space per block is used to store table data. This setting for PCTUSED reduces the number of data blocks required to store the table's data, decreases the average number of data blocks to scan for queries, thereby increasing the performance of queries.

Setting Storage Parameters

You can set storage parameters for the following types of logical storage structures:

- tablespaces (any segment in the tablespace)
- tables, clusters, snapshots, and snapshot logs (data segments)
- indexes (index segments)
- rollback segments

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 listed below, along with their system defaults. (Some defaults are operating system specific; see your installation or user's guide.)



INITIAL

The size, in bytes, of the first extent allocated when a segment is created.

Default: 5 data blocks

Minimum: 2 data blocks

Maximum: operating system specific

Although the default system value is given in data blocks, use bytes to set a value for this parameter. You can use the abbreviations K and M to indicate kilobytes and megabytes. The value is rounded up to the next multiple of the data block size, as determined by the parameter DB_BLOCK_SIZE.

For example, if the data block size of a database is 2048 bytes, then the system default for the INITIAL storage parameter of tablespaces is 10240 bytes. If you create a tablespace in this database and specify its default storage parameter INITIAL as 20000 (bytes), ORACLE automatically rounds this value up to 20480 (10 data blocks).

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 $(1 + \text{PCTINCREASE} / 100)$.

Default: 5 data blocks

Minimum: 1 data block

Maximum: operating system specific

As with INITIAL, although the default system value is given in data blocks, use bytes to set a value for this parameter. You can use the abbreviations K and M to indicate kilobytes and megabytes. The value is rounded up to the next multiple of the data block size, as determined by the parameter DB_BLOCK_SIZE.

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: operating system specific

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: operating system specific If MINEXTENTS is greater than 1, then the speafied number of incremental extents are allocated at creation time using the values INITIAL, NEXT, and PCTINCREASE. Note: The default and minimum values of MINEXTENTS for a rollback segment are always 2. You might want to increase the value for MINEXTENTS when you are creating a rollback segment or when your database is fragmented and you want to guarantee that you have enough space to load all the data for one table.
PCTINCREASE	The percent 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 $1 + \text{PCTINCREASE} / 100$, 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 Note: PCTINCREASE is always 0 for rollback segments. PCTINCREASE cannot be specified for rollback segments. By using PCTINCREASE correctly, you can reduce the fragmentation of a segment by enlarging incremental extents and reducing the number of extents that need to be allocated for the segment.

The segment contains a few large extents, rather than many smaller extents.

If you change PCTINCREASE for a segment, the current value of NEXT for that segment does not change; only future values of NEXT are affected.

INITRANS

Reserves a pre-allocated amount of space for an initial number of transaction entries to access rows in the data block concurrently. Space is reserved in the headers of all data blocks in the associated data or index segment. The default value is 1 for tables (because $1 * 23 \text{ bytes} = 23 \text{ bytes}$) and 2 for clusters and indexes.

MAXTRANS

As multiple transactions concurrently access the rows of the same data block, space is allocated for each 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 is used to limit 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 your installation or user's guide for information.

If MAXTRANS is too low, transactions blocked by this limit must wait until other transactions complete and free transaction entry space. For example, if MAXTRANS is 3 and a fourth concurrent transaction attempts to access a block already being accessed by three active transactions, the statement in the fourth transaction does not continue until one of the previous three transactions commits or rolls back.

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 like 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, both INITRANS and MAXTRANS 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 pre-allocating 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 any necessary data blocks.

Setting Default Storage Parameters for Segments in a Tablespace

Default storage parameters can be set for each tablespace of a database. These default options are used when a segment is created in a tablespace and no storage parameters are explicitly set for the extents of the new segment. See “Creating Additional Tablespaces” on page 7-5.

Setting Storage Parameters for Data Segments

You can set the storage parameters for the data segment of a non-clustered 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, and only if the index’s table already contains rows when the index is created.

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.

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 only PCTINCREASE is altered, the current value of NEXT is recomputed using the formula:

$$\text{NEXT} = \text{NEXT} * (1 + \text{new PCTINCREASE}/100)$$

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. an ALTER TABLE/CLUSTER/SNAPSHOT/SNAPSHOT LOG/INDEX/ROLLBACK SEGMENT statement
2. a CREATE TABLE/CLUSTER/SNAPSHOT/SNAPSHOT LOG/ROLLBACK SEGMENT statement.
3. an ALTER TABLESPACE statement.
4. a CREATE TABLESPACE statement.

Any storage parameter specified at the object level overrides the corresponding option set at the tablespace level. When storage parameters are not explicitly set at the object level, they default to those at the tablespace level. When storage parameters are not set at the tablespace level, ORACLE 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:

Extent #	Extent Size	Value for NEXT
1	100K or 50 blocks	100K
2	100K or 50 blocks	$\text{CEIL}(100\text{K} \times 1.5) = 150\text{K}$
3	150K or 75 blocks	$\text{ceil}(150\text{K} \times 1.5) = 228\text{K}$
4	228K or 114 blocks	$\text{ceil}(228\text{K} \times 1.5) = 342\text{K}$
5	342K or 171 blocks	$\text{ceil}(342\text{K} \times 1.5) = 516\text{K}$

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 $(500\text{K} \times 1.5) = 750\text{K}$, and so on.

Understanding Space Use of Datatypes

In creating tables and other structures, you need to know how much space they will require. Each datatype has different space requirements, as explained in the following sections.

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. (See Appendix C for more information about NLS and support for different character sets.)

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 is specified for the CHAR column; the default is 1 byte.

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 2000 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 -1×10^{-130} to $-9.99...9 \times 10^{125}$ (with up to 38 significant digits), and zero. 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. If precision is not specified, the column stores values as given. If no scale is specified, the scale defaults to zero.

ORACLE guarantees portability of numbers with a precision equal to or less than 38 digits. You can specify a scale and no precision:

```
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 time. 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.

RAW and LONG RAW Datatypes

RAW is equivalent to VARCHAR2, and LONG RAW to LONG, except that SQL*Net (which connects users' sessions to the database instance) and the Export and Import utilities do not perform character conversion when transmitting RAW or LONG RAW data. In contrast, SQL*Net and Export/Import automatically convert CHAR, VARCHAR2, and LONG data between the database character set and the user session character set (set by the NLS_LANGUAGE parameter of the ALTER SESSION command) if the two character sets are different.

LONG RAW data cannot be indexed, while RAW data can be indexed.

ROWIDs and the ROWID Datatype

Every row in a non-clustered 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). In the case of clustered tables, rows in different tables that are in the same data block can have the same ROWID.

Each table in an ORACLE database internally has a *pseudo-column* named ROWID; this pseudo-column 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.row.file*. Here, *block* is the data block that contains the row, relative to its data file; *row* is the row in the block; and *file* is the data file that contains the row. A row's assigned ROWID remains unchanged in most cases; exceptions are when the row is exported and imported (using the `IMPORT` and `EXPORT` utilities) and, for clustered tables only, when values in the cluster key columns of a row change (although ORACLE maintains a marker at the old ROWID, pointing to the row's new location). 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.

You cannot set values of the pseudo-column ROWID; they are used internally by ORACLE for various operations. Although you can reference them, ROWIDS in this pseudo-column are not stored in the database, nor are they database data.

MLSLABEL Datatype

Trusted ORACLE provides one special datatype, called `MLSLABEL`. You can declare columns of this datatype in standard ORACLE, as well as Trusted ORACLE, for compatibility with Trusted ORACLE applications.

The `MLSLABEL` datatype stores a variable length tag (one to five bytes) that represents a binary label in the data dictionary. The `ALL_LABELS` data dictionary view lists all of the labels ever stored in the database. For more information, see the *Trusted ORACLE7 Server Administrator's Guide*.



Summary of ORACLE Datatypes

Table 8-1 summarizes important information about each ORACLE datatype.

TABLE 8-1
Summary of
ORACLE
Datatype
Information

Datatype	Description	Column Length (bytes)
CHAR (<i>size</i>)	Fixed length character data of length <i>size</i> .	Fixed for every row in the table (with trailing spaces); maximum size is 255 bytes per row, default size is one byte per row. Consider the character set that is used before setting <i>size</i> . (Are you using a one or two byte character set?)
VARCHAR2 (<i>size</i>)	Variable length character data. A maximum <i>size</i> must be specified.	Variable for each row, up to 2000 bytes per row. Consider the character set that is used before setting <i>size</i> . (Are you using a one or two byte character set?)
NUMBER (<i>p</i> , <i>s</i>)	Variable length numeric data. Maximum precision <i>p</i> and/or scale <i>s</i> is 38.	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 (<i>size</i>)	Variable length raw binary data. A maximum <i>size</i> must be specified.	Variable for each row in the table, up to 2000 bytes per row.
LONG RAW	Variable length raw binary data.	Variable for each row in the table, up to $2^{31} - 1$ bytes, or two gigabytes, per row.
ROWID	Binary data representing row addresses.	Fixed at six bytes for each row in the table.
MLSLABEL	Variable length binary data representing OS labels.	Variable for each row in the table, ranging from two to five bytes per row.

Managing Tables

This section explains how to plan, create, alter, and drop tables.

Guidelines for Managing Tables

Use the following guidelines to make managing tables as easy as possible.

Design Tables Before Creation

Usually, the application developer is responsible for designing the elements of an application, including the tables. A DBA is 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:

- 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 "Managing the Space Usage of Data Blocks" on page 8-2.)

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 "Setting Storage Parameters" on page 8-5.)

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. (See "Assigning Tablespace Quotas" on page 11-9.)

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 data files.
- 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.

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.

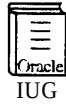
For example, assume that you estimate the maximum size of a table before creating it. If you then set the storage parameters when you create the table, fewer extents will be allocated for the table's data segment, and all of the table's data will be stored in a relatively contiguous section of disk space; this decreases the time necessary for disk I/O operations involving this table.

The following section explains in detail how to estimate the size of tables. Whether or not you estimate table size before creation, you can explicitly set storage parameters when creating each non-clustered 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 Very Large Tables Large tables (such as those with millions of rows and requiring millions of bytes of space) require special planning and attention. If you have such tables in your database, consider the following recommendations:



Limit the Number of Extents in the Table The number of extents in any one segment is limited by the operating system and the database's block size. (See your installation or user's guide for information.) For example, in most databases with a block size of 2K, a segment can have at most 121 extents. Therefore, if a table is going to grow very large, you must ensure that its extents are large enough so that the segment does not have too many extents.

Note: If a segment is allocated the maximum number of extents and still needs more space, you must export, drop, and re-create (or truncate) the object, increasing its storage parameters, and re-import the data into the object. For an extremely large table, this can take as much as 14 hours, or more.

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. (Note that temporary segments always use the default STORAGE settings for their tablespaces.)

Calculating Space Required by Non-Clustered Tables

The following procedure shows how to estimate the initial amount of space required by a non-clustered table in an ORACLE database. For the steps for estimating the size of clustered tables, see "Calculating Space Required by Clusters" on page 8-43.

Several calculations are required to obtain a final estimate, and several of the constants (indicated by *) provided are operating system specific. Your estimates should not significantly differ from actual values. See



your installation or user's guide for any substantial deviations from the constants provided in the following procedure.

This procedure estimates the initial amount of space required for a table in an ORACLE database. When using these estimates, keep in mind the following points that can affect the accuracy of the estimations:

- The space used by transaction entries and deleted records does not become free immediately after completion due to delayed cleanout.
- Trailing nulls and length bytes are not stored.
- Inserts of, updates to, and deletes of rows, as well as columns larger than a single data block, can cause fragmentation and chained row pieces. Therefore, the following estimates will tend to be lower than the actual space required if significant fragmentation occurs.

Once you calculate a table's size using the following procedure, add about 10 percent additional space to calculate the initial extent size for a working table.

The steps to estimate a table's size areas follows:

1. Calculate the total block header size.
2. Calculate the available data space per data block.
3. Calculate the combined column lengths of the average row.
4. Calculate the total average row size.
5. Calculate the average number of rows that can fit in a data block.
6. Calculate the number of blocks and bytes required for the table.

Each step is explained below.

Step 1: Calculate Total Block Header Size

The space required by the data block header is the sum given by the formulas:

block header, part A = fixed header + variable transaction header

Here:

fixed header*	57 bytes
variable transaction header*	23*I
	I is the value of INITRANS for the table.

block header, part B = table directory + row directory

Here:

table directory* 4
row directory 2*R
 R is the number of rows in the block
 (to be calculated in Step 5).

The block header size is broken up into two separate formulas because Part A is applicable to PCTFREE space, while Part B is not; we will need both the total block header size and Part A of the block header size in Step 2 of this procedure.

total block header = block header, part A + block header, part B

If INITTRANS=1, we can partially solve the previous formulas:

total block header = (57 + 23) + (4 + 2R) bytes
 = 80 + (4 + 2R) bytes

Step 2: Calculate Available Data Space Per Data Block

The space reserved in each data block for data, as specified by PCTFREE, is calculated as a percentage of the block size minus the block header:

available
data = (block size - total block header) -
space ((block size - block header, part A)*(PCTFREE/100))

The block size of a database is set during database creation; you can view it using the SQL*DBA command SHOW:

SHOW PARAMETERS db_block_size;



Utilities

Note: For more information about the SQL*DBA command SHOW, see the *ORACLE7 Server Utilities User's Guide*.

Assuming a data block size of 2K and PCTFREE=10 for a given non-clustered table, the total space for new data in data blocks allocated for the table is:

available
data = (2048 - (80 + (4 + 2R)) -
space ((2048 - 80) * (10/100) bytes
 = (1964 - 2R) - (1964 * 0.1) bytes
 = (1964 - 2R - 196) bytes
 = (1768 - 2R) bytes

Step 3: Calculate Combined Data Space

Calculate the space required by the values of an average row using the following information:

- the number of columns in the table definition
- the datatypes used for each column
- the average value size for variable length columns

If a test environment similar to the proposed production database is available, you can use SQL to calculate the combined data space for an average row in a table using the following query:

```
SELECT AVG(NVL(VSIZE (col1), 0)) +
       AVG(NVL(VSIZE (col2), 0)) +
       . . . +
       AVG(NVL(VSIZE (coln), 0)) "SPACE OF AVG ROW"
FROM test_table_name;
```

Here, *col1*, *col2*, ..., *coln* are the column names of the table, and *test_table_name* is the name of the test table being evaluated. This formula assumes that columns containing nulls are not trailing columns, so a column length of 1 is assumed. (The column length of a null in a trailing column is 0.)

For example, assume that you have a test table named T created with the following statement:

```
CREATE TABLE t (
  a  CHAR (10),
  b  DATE,
  c  NUMBER (10, 2));
```

To determine the space required by the average row in table T, issue the following query:

```
SELECT AVG(NVL(VSIZE(A),0)) +
       AVG(NVL(VSIZE(B),0)) +
       AVG(NVL(VSIZE(C),0)) "SPACE OF AVG ROW"
FROM t;
```

Alternatively, if a test database is not present, you can estimate the combined data space for an average row in a table. This is done by examining the datatype of each column in a table. If the column is of a fixed-length datatype, the value for the column in each row of the table is fixed. For variable-length columns, you must determine the average column value and determine how much space is required to hold this typical value. Table 8-1 provides a summary of datatype information to reference for this procedure.

For example, suppose you would like to calculate the combined column lengths of an average row in the table T. The column lengths of the first two columns are simple to determine because each column is of a fixed-length datatype. Each row in table T requires 10 bytes for

values in column A (assuming a single byte character set is being used), and 7 bytes for values in column B.

The space required by values in the third column, C, can vary for each row in the table because the NUMBER datatype is a variable-length datatype. By estimating your proposed data, you might determine that the typical value in this column would have a precision of 8 (scale is irrelevant), and is a positive value. The typical value in column C requires approximately:

$$\begin{aligned}\text{average length column C} &= (\text{average } p/2 + 1) \text{ bytes} \\ &= (8/2 + 1) \text{ bytes} \\ &= 5 \text{ bytes}\end{aligned}$$

In summary, the combined space required for the columns of an average row in table T is:

$$\begin{aligned}D \text{ (data space/average row)} &= (A + B + C) \\ &= (10 + 7 + 5) \text{ bytes} \\ &= 22 \text{ bytes}\end{aligned}$$

Step 4: Calculate Total Average Row Size

You can calculate the minimum amount of space required by a row in a non-clustered table with the following equation:

$$\text{bytes/row} = \text{row header} + F + V + D$$

where:

row header*	3 bytes per row of a non-clustered table.
F	Total length bytes of all columns that store 250 bytes or less. The number of length bytes required by each column of this type is 1 byte.
V	Total length bytes of all columns that store more than 250 bytes. The number of length bytes required by each column of this type is 3 bytes.
D	Combined data space of all columns in average row (from Step 3).

For example, the total average row size of table T used in Step 3 is:

$$\begin{aligned}\text{avg. row size, table T} &= (3 + (1 * 3) + (3 * 0) + 22) \text{ bytes} \\ &= 28 \text{ bytes}\end{aligned}$$

The absolute minimum row size of a non-clustered row is 9 bytes*. Therefore, if your calculated value for a table's total average row size is less than these absolute minimum row sizes, use the minimum value as the average row size in subsequent calculations.

Step 5: Calculate Average Rows Per Block

After the average row size is determined, you can calculate the average number of rows that can fit into a data block:

$$R \text{ (avg. \# of rows/block)} = \text{available space} / \text{average row size}$$

The available space was calculated in Step 2 and the average row size was calculated in Step 4. Continuing the example of the previous steps, the average number of rows in table T (a non-clustered table) that can fit into a data block is:

$$\begin{aligned} R \text{ rows/block} &= (1768 \text{ bytes} - 2R \text{ bytes} * \text{rows/block}) / 28 \text{ bytes} \\ 28R \text{ bytes} * \text{rows/block} &= 1768 \text{ bytes} - 2R \text{ bytes} * \text{rows/block} \\ 30R \text{ bytes} * \text{rows/block} &= 1768 \text{ bytes} \\ R &= 58 \text{ rows/block} \end{aligned}$$

Step 6: Calculate Number of Blocks and Bytes

Once you know the number of rows that can fit inside the available space of a data block, you can calculate the number of blocks required to hold the proposed table:

$$\# \text{ blocks for table} = \# \text{ rows} / R \text{ rows/block}$$

For example, if you estimate that table T will have 10000 rows:

$$\begin{aligned} \# \text{ blocks for table T} &= 10000 \text{ rows} / 58 \text{ rows/block} \\ &= 173 \text{ blocks} \end{aligned}$$

To calculate the same space value in bytes, multiply the above result by the data block size of the corresponding database. For example, the block size in this database is 2048 bytes:

$$\# \text{ bytes for table} = \# \text{ blocks for table} * 2048 \text{ bytes/block}$$

Continuing with the example of table T:

$$\begin{aligned} \# \text{ bytes for table T} &= 173 \text{ blocks} * 2048 \text{ bytes/block} \\ &= 354304 \text{ bytes (346K)} \end{aligned}$$

In summary, remember that this procedure provides a reasonable estimate of a table's size, not an exact number of blocks or bytes. Once you have estimated the size of a table, you can use this information when specifying the INITIAL storage parameter (size of the table's initial extent) in your corresponding CREATE TABLE statement.

Space Requirements for Tables in Use

Once a table is created and in use, the space required by the table is usually higher than the estimate given by the previous section. More space is required due to the method by which ORACLE manages free space in the database.

Creating Tables

Create tables using the SQL command CREATE TABLE. For example, if the user SCOTT issues the following statement, he creates a

non-clustered 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.

Privileges Required to Create a Table

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. (See Chapter 12 for more information about system privileges and Chapter 11 for more information about tablespace quotas.)

Altering Tables

A table in an ORACLE database can be altered for the following reasons:

- to add one or more new columns to the table
- to add one or more integrity constraints to 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, etc.)

- 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; 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 (INITTRANS, MAXTRANS) of a table, note that a new setting for INITTRANS only applies 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 (e.g., 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.

Alter a table using the SQL command ALTER TABLE. For example, the following statement alters the EMP table:

```
ALTER TABLE emp
  PCTFREE 30
  PCTUSED 60;
```

You should understand the following implications of altering a table:

- 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 page 8-75 for information about how ORACLE manages such dependencies.

Privileges Required to
Alter a Table

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.

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. For more information about this parameter of the ALTER TABLE command, see the *ORACLE7 Parallel Server Administrator's Guide*.

Dropping Tables

To drop a table that is no longer needed, use the SQL command DROP TABLE. For example, 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 below:

```
DROP TABLE emp CASCADE CONSTRAINTS;
```

Before dropping a table, consider the following effects:



- 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 page 8-75 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 non-clustered table that is dropped are returned to the free space of the tablespace and can be used by any other object requiring new extents.
- All rows corresponding to a clustered table are deleted from the blocks of the cluster.

Privileges Required to
Drop a Table

To drop a table, the table must be contained in your schema or you must have the DROP ANY TABLE system privilege to drop the table.

Managing Views

The following sections explain how to create, replace, and drop views using several SQL commands.

Creating Views

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 `ORDER BY` or `FOR UPDATE` clauses. 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 via 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; because rows cannot be inserted into or updated in a view defined with a query that contains a join, the `CHECK OPTION` is not meaningful for the `DIVISION1_STAFF` 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 quotation marks in case the columns of the base object were originally entered with quotation marks and require them for the query to be syntactically correct. Quotation marks are always included because the data dictionary does not store whether the user used quotation marks to specify the identifier in the original DDL statement that created the column and because it cannot reliably deduce from the column name itself whether or not the quotes are required. Although the quotation marks are unnecessary when a column name contains only numbers and capital letters and starts with a capital letter, they are harmless in these cases.

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
```

Note: ORACLE does not expand wildcards in views created with errors (see the next section). When the view is eventually compiled without errors, ORACLE expands wildcards in the defining query.

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 “Managing Object Dependencies” on page 8-75.

Privileges Required to Create a View

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.

Replacing Views

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. When a view is dropped, all grants of corresponding object privileges are revoked from roles and users. After the view is re-created, privileges must be regranted.
- You can redefine the view with a CREATE VIEW statement that contains the OR REPLACE 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 page 8-75 for more information on how ORACLE manages such dependencies.

Privileges Required to Replace a View

To replace a view, you must have all the privileges required to drop and create a view.

Dropping Views

Drop a view using the SQL command DROP VIEW. For example, the following statement drops a view named SALES_STAFF:

```
DROP VIEW sales_ staff;
```

Privileges Required to Drop a View

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.

Managing Sequences

The following sections explain how to create, alter, and drop sequences using SQL commands.

Creating Sequences

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. ORACLE might also skip cached sequence numbers after an export and import; see the *ORACLE7 Server Utilities User's Guide* for details.

The ORACLE Parallel Server has additional effects on cached sequence numbers; see the *ORACLE7 Parallel Server Administrator's Guide* for more information. For performance information on caching sequence numbers, see the *ORACLE7 Server Application Developer's Guide*.

Privileges Required to Create a Sequence

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.

Altering Sequences

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.

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;
```

Privileges Required to Alter a Sequence

To alter a sequence, your schema must contain the sequence, or you must have the ALTER ANY SEQUENCE system privilege.

Initialization Parameters Affecting Sequences

The initialization parameter SEQUENCE_CACHE_ENTRIES sets the number of sequences that maybe 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

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.

Privileges Required to Drop a Sequence

You can drop any sequence in your schema. To drop a sequence in another schema, you must have the `DROP ANY SEQUENCE` system privilege.

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.

The following sections explain how to create and drop synonyms using SQL commands.

Creating Synonyms

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;
```

Privileges Required to Create a Synonym

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.

Dropping Synonyms

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 “Managing Object Dependencies” on page 8-75.

Privileges Required to Drop a Synonym

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.

Managing Indexes

The following sections explain how to create, alter, and drop indexes using SQL commands, and provide guidelines on working with indexes.

Guidelines for Managing Indexes

The following sections give some guidelines to follow when managing indexes.



AppDev

See the *ORACLE7 Server Application Developer’s Guide* for performance implications of index creation.

Create Indexes After Inserting Table Data



For an index for a table, you should create the index after data has been inserted or loaded (by 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.

For an index for a cluster, you must create a cluster’s index before any data can be inserted into the cluster.

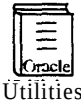
When an index is created on a table that already has data, ORACLE must use sort space to create the index. ORACLE uses the sort space in memory allocated for the creator of the index (the amount per user is determined by the initialization parameter SORT_AREA_SIZE), but

must also swap sort information to and from temporary segments allocated on behalf of the index creation.



If the index is extremely large, it may be beneficial to perform the following steps:

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 respecify the creator's temporary segment tablespace, if desired.



Note: 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 the *ORACLE7 Server Utilities User's Guide* for more information.

Index the Correct Tables and Columns

LONG and LONG RAW columns cannot be indexed. Also, the size of a single index entry cannot exceed roughly one-third the data block size. See "Calculating Space for Indexes" on page 8-35 to determine if an index violates this limit.

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 altered. 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 tradeoff between speed of retrieving data from a table and 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 can 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. For more information about setting these parameters, see "Setting Storage Parameters" on page 8-5.

Specify Data 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 in another index block, which is chained to the head block for the index leaf. 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.

Note: PCTUSED cannot be specified for indexes. See “Managing the Space Usage of Data Blocks” on page 8-2 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 maybe 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.

Estimate Index Size and Set Storage Parameters

The reasons for estimating an index’s size before creation are identical to those for estimating table size. See “Estimate Table Size and Set Storage Parameters” on page 8-16 if you are not familiar with this justification. “Calculating Space for Indexes” on page 8-35 explains how to estimate the size of indexes.

As with tables, you can explicitly set storage parameters when creating an index. If you explicitly set the storage parameters for an index, try to store the index’s data in a small number of large extents rather than a large number of small extents.

Any storage parameter not explicitly set for an index automatically uses the corresponding default storage parameter set for the containing tablespace. If you do not specify storage parameters when creating an index, ORACLE allocates extents to index segments based on the total size of the index:

- If the table being indexed contains no rows when the index is created, extent sizes are allocated according to the default storage parameters of the containing tablespace.

- An index smaller than or equal to 25 blocks is allocated in one extent by default.
- For indexes larger than 25 blocks, the default sizes of the initial and incremental extents are one-fifth of the total index size.

For specific information about storage parameters, see “Set Storage Parameters” on page 8-5.

Consider Indexes 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 or not 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.

Calculating Space for Indexes



The following procedure demonstrates how to estimate the initial amount of space required by an index. Several calculations are required to obtain a final estimate. Also, several of the constants (indicated by *) provided are operating system specific, yet should provide reasonable results that will not differ significantly from actual values. See your installation or user's guide for any dramatic deviations from the constants provided in the following procedure.

The calculations in the procedure rely on average column lengths of the columns that constitute an index; therefore, if column lengths in each row of a table are relatively constant with respect to the indexed columns, the estimates calculated by the following procedure are more accurate.

The steps to estimate an index's size areas follows:

1. Calculate the total block header size.
2. Calculate the available data space per data block.
3. Calculate the combined column lengths of an average index value.
4. Calculate the total average index value size.
5. Calculate the number of blocks and bytes required for the index.

Each step is explained below.

Step 1: Calculate the Total Block Header Size

The space required by the data block header of a block to contain index data is given by the formula:

block header size = fixed header + variable transaction header

where:

fixed header*	113 bytes
variable	23*I
transaction	I is the value of INITTRANS for the index.
header*	

If INITTRANS = 2 (the default for indexes), the previous formulas can be simplified:

$$\begin{aligned}\text{block header} &= 113 + (23 \times 2) \text{ bytes} \\ &= 159 \text{ bytes}\end{aligned}$$

Step 2: Calculate Available Data Space Per Data Block

The space reserved in each data block for index data, as specified by PCTFREE, is calculated as a percentage of the block size minus the block header

$$\begin{aligned}\text{available} & \\ \text{data} &= (\text{block size} - \text{block header}) - \\ \text{space} &((\text{block size} - \text{block header}) * (\text{PCTFREE}/100))\end{aligned}$$

The block size of a database is set during database creation and can be determined using the SQL*DBA command SHOW, if necessary:

```
SHOW PARAMETERS db_block_size;
```



Note: For more information about the SQL*DBA command SHOW, see the *ORACLE7 Server Utilities User's Guide*.

If the data block size is 2K and PCTFREE=10 for a given index, the total space for new data in data blocks allocated for the index is:

$$\begin{aligned}\text{available} & \\ \text{data} &= (2048 \text{ bytes} - 159 \text{ bytes}) - \\ \text{space} &((2048 \text{ bytes} - 159 \text{ bytes}) * (10/100)) \\ &= (1889 \text{ bytes}) - (1889 \text{ bytes} * 0.1) \\ &= 1889 \text{ bytes} - 188.9 \text{ bytes} \\ &= 1700.1 \text{ bytes}\end{aligned}$$

Step 3: Calculate Combined Column Lengths of an Index Value

The space required by the average value of an index must be calculated before you can complete Step 4, calculating the total row size. This step is identical to Step 3 in the procedure for calculating table size (page 8-19), except you only need to calculate the average combined column lengths of the columns in the index.

Step 4: Calculate Total Average Index Value Size

Once you have calculated the combined column length of an average index entry, you can calculate the total average entry size according to the following formula:

$$\text{bytes/entry} = \text{entry header} + \text{ROWID length} + F + V + D$$

where:

entry header	2 bytes
ROWID length	6 bytes
F	Total length bytes of all columns that store 128 bytes or less. The number of length bytes required by each column of this type is 1 byte.
V	Total length bytes of all columns that store more than 128 bytes. The number of length bytes required by each column of this type is 3 bytes.
D	Combined data space of all index columns (from Step 3).

For example, given that D is calculated to be 22 bytes and that the index is comprised of three CHAR(10) columns, the total average entry size of the index is:

$$\begin{aligned} \text{avg. entry size} &= 1 + 6 + (1 \times 3) + (3 \times 0) + 22 \text{ bytes} \\ &= 32 \text{ bytes} \end{aligned}$$

Step 5: Calculate Number of Blocks and Bytes

Calculate the number of blocks required to store the index using the following formula:

blocks for index =

$$1.05 * \frac{\text{\# not null rows} * \text{avg. entry size}}{(\text{FLOOR}(\text{avail. data space} / \text{avg. entry size})) * (\text{avg. entry size})}$$

Note: The additional 5% added to this result (via the multiplication factor of 1.05) accounts for the extra space required for branch blocks of the index.

For example, continuing with the previous example, and assuming you estimate that indexed table will have 10000 rows that contain non-null values in the columns that constitute the index:

blocks for index =

$$1.05 * \frac{10000 * 32 \text{ bytes}}{(\text{FLOOR}(1700 \text{ bytes} / 32 \text{ bytes})) * (32 \text{ bytes})}$$

This results in 198 blocks. The number of bytes can be calculated by multiplying the number of blocks by the data block size.

Remember that this procedure provides a reasonable estimate of an index's size, not an exact number of blocks or bytes. Once you have estimated the size of a index, you can use this information when specifying the INITIAL storage parameter (size of the index's initial extent) in your corresponding CREATE INDEX statement.

Temporary Space Required for Index Creation

When creating an index for a loaded table, temporary segments are created to sort the index. The amount of space required to sort an index varies, but can be up to 110% of the size of the index.

Note: Temporary space is not required if the NOSORT option is included in the CREATE INDEX command. However, you cannot specify this option when creating a cluster index.

Creating Indexes

An index can be created for a table to improve the performance of queries issued against the corresponding table. An index can also be created for a cluster.

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.

Note: 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.

In both types of index creation, if you do not specify the storage options for an index, they are automatically set to the default storage options of the containing tablespace.

Creating an Index Associated with a constraint

Similarly to the above example, 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. For example, 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

Indexes are created explicitly (outside of integrity constraints) using the SQL command `CREATE INDEX`. For example, 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.

Privileges Required to Create an Index

To create a new index you must own or have the `INDEX` object privilege for the corresponding table. The schema that contains the index must also have a quota for the tablespace intended to contain the index, or the `UNLIMITED TABLESPACE` system privilege. To create an index in another user's schema, you must have the `CREATE ANY INDEX` system privilege.

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.

Altering an Index

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` only applies to

data blocks subsequently allocated, while a new setting for MAXTRANS applies to all blocks (already 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;
```

Privileges Required to
Alter an Index

To alter an index, your schema must contain the index or you must have the ALTER ANY INDEX system privilege.

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 validating the index's structure (see "Analyzing Tables, Indexes, and Clusters" on page 8-62) 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: validating the index, checking PCT_USED, and dropping and re-creating 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 re-creating it.

Dropping Indexes

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. (See “Analyzing Tables, Indexes, and Clusters” on page 8-62.)
- 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. The following sections explain how to drop both kinds of indexes.

Note: If a table is dropped, all associated indexes are dropped automatically.

Dropping an Index
Associated with an
Integrity Constraint

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. See “Managing Integrity Constraints” on page 8-70.

Dropping an Index
Created Explicitly

Drop an index with the SQL command DROP INDEX. For example, to drop the EMP_ENAME index, enter the following statement

```
DROP INDEX emp_ename;
```

Privileges Required to
Drop An Index

To drop an index, the index must be contained in your schema or you must have the DROP ANY INDEX system privilege.

Managing Clusters, Clustered Tables, and Cluster Indexes

The following sections explain how to create, alter, and drop clusters, clustered tables, and cluster indexes. For additional information on managing hash clusters, see “Managing Hash Clusters and Clustered Tables” on page 8-54.

Guidelines for Managing Clusters

The following sections give some guidelines for creating clusters.

Choose Appropriate
Tables to 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; see “Index the Correct Tables and Columns” on page 8-33 for these guidelines.

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`.

Specify How Data Block Space Is to Be Used

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 “Managing the Space Usage of Data Blocks” on page 8-2 for more information and guidelines for setting `PCTFREE` and `PCTUSED` parameters.

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 in 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.

Use the procedure in "Calculating Space Required by Clusters" on page 8-43 to calculate SIZE for a cluster.

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

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

The reasons for estimating a cluster's size before creation are identical to those for estimating table size. See "Estimate Table Size and Set Storage Parameters" on page 8-16.

Use the procedure in the following section to calculate the space required by a cluster.

Calculating Space Required by Clusters



The following procedure demonstrates how to estimate the initial amount of space required by a set of tables in a cluster. Several calculations are required to obtain a final estimate, and several of the constants (indicated by *) provided are operating system specific. Your estimates should not significantly differ from actual values. See your installation or user's guide for any substantial deviations from the constants provided in the following procedure.

Note that this procedure estimates only the initial amount of space required for a cluster. When using these estimates, keep in mind the following points that can affect the accuracy of the estimations:

- The space used by transaction entries and deleted records does not become free immediately after completion due to delayed cleanout.
- Trailing nulls are not stored, nor is a length byte.
- Inserts of, updates to, and deletes of rows, as well as tables containing columns larger than a single data block can cause fragmentation and chained row pieces. Therefore, the following estimates may tend to be lower than the actual space required if significant fragmentation occurs.

Once you calculate a table's size using the following procedure, you should add about 10 to 20 percent additional space to calculate the initial extent size for a working table.

The steps to estimate a cluster's size are as follows:

1. Calculate the total block header size.
2. Calculate the available data space per data block.
3. Calculate the combined column lengths of the average rows per cluster key.
4. Calculate the average row size of all clustered tables.
5. Calculate the average cluster block size.
6. Calculate the total number of blocks required for the cluster.

Each step is explained below.

Step 1: Calculate Total Block Header Size

The following formula returns the amount of space required by the data block header

$$\text{block header} = \text{fixed header} + \text{variable transaction header} + \text{table directory} + \text{row directory}$$

where:

fixed header*	57 bytes
variable transaction header*	23*I <i>I</i> is the value of INITRANS for the table.
table directory*	4*N + 1 <i>N</i> is the number of tables in the data segment.

row directory* $2 \times R$
 R is the number of rows in the block. R will be estimated and the space required to hold the row directory information included in Step 5.

IF INITTRANS = 2, the previous formula can be partially solved:

$$\begin{aligned} \text{block header} &= 57 + 46 + (4N + 1) + 2R \text{ bytes} \\ &= 104 + 4N + 2R \text{ bytes} \end{aligned}$$

Step 2: Calculate Available Data Space Per Data Block

The space reserved in each data block for data, as specified by PCTFREE, is calculated as the block size minus the block header:

$$\begin{aligned} \text{available} \\ \text{data} \quad &= (\text{block size} - \text{total block header}) \\ \text{space} \end{aligned}$$

The block size of a database is set during database creation and can be determined using the SQL*DBA command SHOW:

```
SHOW PARAMETERS db_block_size;
```



Utilities

Note: For more information about the SQL*DBA command SHOW, see the *ORACLE7 Server Utilities User's Guide*.

Assuming a data block size of 2K, the total space for new data in data blocks allocated for the cluster is as follows:

$$\begin{aligned} \text{available} \\ \text{data} \quad &= (2048 - (104 + 4N + 2R)) \\ \text{space} \quad &= (1944 - 4N - 2R) \text{ bytes} \\ &= (1944 - 4N - 2R) \text{ bytes} \end{aligned}$$

Assuming there are two tables in the clusters (that is, $N = 2$), the formula simplifies to:

$$\begin{aligned} \text{available} \\ \text{data} \quad &= 1944 - (4 \times 2) - 2R \text{ bytes} \\ \text{space} \quad &= 19362 - 2R \text{ bytes} \end{aligned}$$

Step 3: Calculate Space Required by a Row

Use Step 3 in the procedure discussed in “Calculating Space Required by Non-Clustered Tables” on page 8-19 to calculate this number. Make note of the following caveats:

- Calculate the data space required by an average row for each table in the cluster. For example, in a cluster that contains tables T1 and T2, calculate the average row size for both tables.

- Do not include the space required by the cluster key in any of the above calculations. However, make note of the space required to store an average cluster key value for Step 5. For example, calculate the data space required by an average row in table T1, not including the space required to store the cluster key.
- Do not include any space required by the row header (that is, the length bytes for each column); this space is accounted for in the next step.

For example, assume two clustered tables are created with the following statements:

```
CREATE TABLE t1 (a CHAR(10), b DATE, c NUMBER (10,2))
CLUSTER t1_t2 (c);
```

```
CREATE TABLE t2 (c NUMBER (10,2), d CHAR(10))
CLUSTER t1_t2 (c);
```

Notice that the cluster key is column C in each table.

Considering these example tables, the space required for an average row (D_1) of table T1 and the space required for an average row (D_2) of table T2 is:

$$\begin{aligned} D_1 (\text{space/average row}) &= (a + b) \\ &= (10 + 7) \text{ bytes} \\ &= 17 \text{ bytes} \end{aligned}$$

$$\begin{aligned} D_2 (\text{space/average row}) &= (d) \\ &= 10 \text{ bytes} \end{aligned}$$

Step 4: Calculate Total Average Row Size

You can calculate the minimum amount of space required by a row in a clustered table according to the following equation:

$$S_n \text{ bytes/row} = \text{row header} + F_n + V_n + D_n$$

where:

row header*	4 bytes per row of a clustered table.
F_n	Total length bytes of all columns in table n that store 250 bytes or less. The number of length bytes required by each column of this type is 1 byte.
V_n	Total length bytes of all columns in table n that store more than 250 bytes. The number of length bytes required by each column of this type is 3 bytes.
D_n	Combined data space of all columns in table n (from Step 3).

Note: Do not include the column length for the cluster key in variables F or V for any table in the cluster. This space is accounted for in Step 5.

For example, the total average row size of the clustered tables T1 and T2 are:

$$S_1 = (4 + (1 * 2) + (3 * 0) + 17) \text{ bytes} \\ = 23 \text{ bytes}$$

$$S_2 = (4 + (1 * 1) + (3 * 0) + 10) \text{ bytes} \\ = 15 \text{ bytes}$$

Note: The absolute minimum row size of a clustered row is 10 bytes, and is operating system specific. Therefore, if your calculated value for a table's total average row size is less than these absolute minimum row sizes, use the minimum value as the average row size in subsequent calculations.

Step 5: Calculate Average Cluster Block Size

To calculate the average cluster block size, first estimate the average number of rows (for all tables) per cluster key. Once this is known, use the following formula to calculate average cluster block size:

$$\text{avg. cluster block size (bytes)} = \\ ((R_1 * S_1) + (R_2 * S_2) + \dots + (R_n * S_n)) + \text{key header} + C_k + S_k + 2R_t$$

where:

R_n	The average number of rows in table n associated with a cluster key.
S_n	The average row size in table n (see Step 4).
key header*	19
C_k	Column length for the cluster key.

S_k	Space required to store average cluster key value.
R_t	Total number of rows associated with an average cluster key ($R_1 + R_2 + \dots + R_n$). This accounts for the space required in the data block header for each row in the block.

For example, consider the cluster that contains tables T1 and T2. An average cluster key has one row per table T1 and 20 rows per table T2. Also, the cluster key is of datatype NUMBER (column length is 1 byte), and the average number is 4 digits (3 bytes). Considering this information and the previous results, the average cluster key size is:

$$\begin{aligned} \text{SIZE} &= ((1 * 23) + (20 * 15) + 19 + 1 + 3 + (2 * 21)) \text{ bytes} \\ &= 388 \text{ bytes} \end{aligned}$$

Specify the estimated SIZE in the SIZE option when you create the cluster with the CREATE CLUSTER command. This specifies the space required to hold an average cluster key and its associated rows; ORACLE uses the value of SIZE to limit the number of cluster keys that can be assigned to any given data block. After estimating an average cluster key SIZE, choose a SIZE somewhat larger than the average expected size to account for the space required for cluster keys on the high side of the estimate.

To estimate the number of cluster keys that will fit in a database block, use the following formula, which uses the value you calculated in Step 2 for available data space, the number of rows associated with an average cluster key (R_t), and SIZE:

$$\begin{aligned} \# \text{ cluster keys} &= \text{FLOOR} (\text{available data space} + 2R_t / \\ \text{per block} &\quad \text{SIZE} + 2R_t) \end{aligned}$$

For example, with SIZE previously calculated as 400 bytes (calculated as 388 earlier in this step and rounded up), R_t estimated at 21, and available space per data block (from Step 2) calculated as 1742 - 2R bytes, the result is as follows:

$$\begin{aligned} \# \text{ cluster keys} &= \text{FLOOR} ((1936 - 2R + 2R_t) / \\ \text{per block} &\quad (400 + 2 * 21)) \\ &= \text{FLOOR} (1936 / 442) \\ &= \text{FLOOR} (4.4) \\ &= 4 \end{aligned}$$

Step 6: Calculate Total Number of Blocks

To calculate the total number of blocks for the cluster, you must estimate the number of cluster keys in the cluster. Once this is estimated, use the following formula to calculate the total number of blocks required for the cluster:

```
# blocks = CEIL(# cluster keys / # cluster keys per block)
```

Note: If you have a test database, you can use statistics generated by the ANALYZE command to determine the number of key values in a cluster key. See “Analyzing Tables, Indexes, and Clusters” on page 8-62.

For example, assume that there is approximately 500 cluster keys in the T1_T2 cluster.

```
# blocks T1_T2 = CEIL (500/3)
                = CEIL (166.7)
                = 167
```

To convert the number of blocks to bytes, multiply the number of blocks by the data block size.

This procedure provides a reasonable **estimation** of a cluster’s size, but not an exact number of blocks or bytes. Once you have estimated the space for a cluster, you can use this information when specifying the INITIAL storage parameter (size of the cluster’s initial extent) in your corresponding CREATE CLUSTER statement.

Space Requirements for Clustered Tables in Use

Once clustered tables are created and in use, the space required by the tables is usually higher than the estimate given by the previous section. More space is required due to the method ORACLE uses to manage free space in the database.

Creating Clusters, Clustered Tables, and Cluster Indexes

A cluster is used to store one or more tables that are frequently joined together in queries. A cluster should not be used to cluster tables that are frequently accessed individually.

Once a cluster is created, tables can be created in the cluster. However, before any rows can be inserted into the clustered tables, a cluster index must be created. The use of clusters does not affect the creation of additional indexes on the clustered tables; they can be created and dropped as usual.

Create a cluster using the SQL command CREATE CLUSTER. For example, 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);

```

A table is created in a cluster using the SQL command CREATE TABLE with the CLUSTER option. For example, 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 is specified in the CREATE TABLE statement; a clustered table can be in a different schema than that which contains the cluster.

A cluster index must be created before any rows can be inserted into any clustered table. For example, 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;

```

The cluster key establishes the relationship of the tables in the cluster. Several storage settings are explicitly specified for the cluster and cluster index.

Privileges Required to Create a Cluster, Clustered Table, and Cluster Index

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.

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.

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.

See Chapter 12 for more information about system privileges and Chapter 11 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, etc.)

Note: See “Managing Tables” and “Managing Clusters, Cluster Tables, and Cluster Indexes” on pages 8-15 and 8-41, respectively, for guidelines on these options.

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 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 SQL command ALTER CLUSTER. For example, the following statement alters the EMP_DEPT cluster:

```
ALTER CLUSTER emp_dept  
  PCTFREE 30  
  PCTUSED 60;
```

Privileges Required to Alter a Cluster

To alter a cluster, your schema must contain the cluster or you must have the ALTER ANY CLUSTER system privilege.

Altering Clustered Tables and Cluster Indexes

You can alter clustered tables using the SQL command ALTER TABLE. However, any data block space parameters, transaction entry parameters, or storage parameters you set in an ALTER TABLE statement for a clustered table are ignored; 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, or add, drop, enable, or disable integrity constraints or triggers for a clustered table. See “Altering Tables” on page 8-23.

You alter cluster indexes exactly as you do other indexes. See “Altering an Index” on page 8-39.

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.



Parallel

You allocate a new extent for a cluster using the SQL command ALTER CLUSTER with the ALLOCATE EXTENT option. For more information about this parameter of the ALTER CLUSTER command, see the *ORACLE7 Parallel Server Administrator's Guide*.

Dropping Clusters, Clustered Tables, and 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.

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 SQL command `DROP TABLE`; see "Dropping Tables" on page 8-25.

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` command with the `INCLUDING TABLES` option. Drop an individual table from a cluster (using the `DROP TABLE` command) only if you want the rest of the cluster to remain.

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. See "Dropping Indexes" on page 8-40.

To drop a cluster that contains no tables, and its cluster index, use the SQL command `DROP CLUSTER`. 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` command, as in:

```
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` command. For 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.

Privileges Required to Drop a Cluster

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.

Managing Hash Clusters and Clustered Tables

The following sections explain how to create, alter, and drop hash clusters and clustered tables using SQL commands. Several examples of these commands are given.

Deciding When to Use Hashing

Storing a table in a hash cluster is an alternative to storing the same table with an index. Hashing is useful in the following situations:

- Most queries are equality queries on the cluster key. For example:

```
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.

Hashing is not advantageous in the following situations:

- Most queries on the table retrieve rows over a range of cluster key values. For example, in full table scans, or queries such as:

```
SELECT . . . WHERE cluster_key < . . . ;
```

In such cases, 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. 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 (thus, its cluster) cannot be predetermined.
- Applications frequently perform full-table scans on the table and the table is sparsely populated. A full-table scan in this situation takes longer under hashing.
- You cannot afford to pre-allocate the space that the hash cluster will eventually need.

In most cases, you should decide (based on the above information) whether to use hashing or indexing. If you decide to use indexing, you should consider whether to store a table individually or as part of a cluster; see “Choose Appropriate Tables to Cluster” on page 8-41.

Note: Even if you decide to use hashing, a table can still have separate indexes on any columns, including the cluster key. See the *ORACLE7 Server Application Developer's Guide* for additional recommendations.



Calculating Space Required by Hash Clusters

As with index clusters, it is important to estimate the storage required for the data in a hash cluster. Use the procedure explained in “Calculating Space Required by Clusters” on page 8-43, with the following additional notes:

- A sub-goal of the procedure is to determine the SIZE of each cluster key. However, for hash clusters, the corresponding sub-goal is to determine the SIZE of each hash key. Therefore, you must consider not only the number of rows per cluster key value, but also the distribution of cluster keys over the hash keys in the cluster.
- In Step 3, make sure to include the space required by the cluster key value. Unlike an index cluster, the cluster key value is stored with each row placed in a hash cluster.
- In Step 5, you are calculating the average hash key size, not cluster key size. Therefore, take into account how many cluster keys map to each hash value. Also, disregard the addition of the space required by the cluster key value, C_k . This value has already been accounted for in Step 3 (see previous item).

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:

```
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 settings for the storage parameters account for this requirement (an initial extent of 100K is allocated to the hash cluster).

Creating Hash Clusters and Clustered Tables

A hash cluster is used to store individual tables or a group of clustered tables that are static and often queried by equality queries. Once a hash cluster is created, tables can be created in the cluster.

A hash cluster is created using the SQL command CREATE CLUSTER. For example, the following statement creates a cluster named TRIAL_CLUSTER that is used to store 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)
  SIZE 2K
  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. 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, Clustered Tables, and Cluster Indexes” on page 8-49.

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 sections provide guidance as well as examples of setting these parameters.

Choosing the Key Choosing the correct cluster key is dependent on the most common types of queries issued against the clustered tables. For example, consider the EMP table in a hash cluster. If queries often select rows by employee number, the EMPNO column should be the cluster key; if queries often select rows by department number, the DEPTNO column should be the cluster key. For hash clusters that contain a single table, the cluster key is typically the entire primary key of the contained table.

The key of a hash cluster (like that of an index cluster) can be a single column or a composite key (multiple column key). A hash cluster with a composite key must use ORACLE’s internal hash function.

Setting HASH IS Only specify the HASH IS parameter 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 to a unique hash value (with no collisions); also see the following sections for information on setting SIZE and HASHKEYS. If the above 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.

To calculate a preliminary value for SIZE, follow the procedures given in “Calculating Space Required by Hash Clusters” on page 8-55. If the

preliminary value for SIZE is small (more than four hash keys can be assigned per data block), you can use this value for SIZE in the CREATE CLUSTER command. However, if the value of SIZE is large (fewer than five hash keys can be assigned per data block), you should also consider the expected frequency of collisions and whether performance of data retrieval or efficiency of space usage is more important to you:

- If the hash cluster does not use the internal hash function (if you specified HASH IS) and you expect little or no collisions, you can set SIZE as calculated above; 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 the following chart:

Available Space per Block/Calc'd SIZE	Setting for SIZE
1	Calc'd SIZE
2	Calc'd SIZE + 15%
3	Calc'd SIZE + 12%
4	Calc'd SIZE + 8%
> 4	Calc'd 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 calculated value for SIZE.

Setting HASHKEYS HASHKEYS should always be set to the number of unique cluster key values rounded up to the next prime number (assuming the guidance in the previous two sections is used). For maximum distribution of rows in a hash cluster, HASHKEYS should always be a prime number.

For example, suppose you cluster the EMP table by DEPTNO, and there are 100 DEPTNOs, with values 10,20,...,1000. Assuming you bypass the internal hash function and you create a cluster with HASHKEYS of 100, then department 10 will hash to 10, department 20 to 20,..., department 110 to 10 (110 mod 100), department 120 to 20, and so on. Notice that there are 10 entries for hash values of 10,20,..., but none for 1,2,..., and so on. As a result, there is a lot of wasted space and possibly a lot of overflow blocks because of collisions. Alternatively, if HASHKEYS is set to 101, then each department number hashes to a unique hash key value.

Examples of Choosing the Key and Setting HASH IS, SIZE, and

HASHKEYS 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, 10001:

```
CREATE CLUSTER emp_cluster ( empno NUMBER)
. . .
SIZE 55
HASH IS empno HASHKEYS 10001;
```

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 precalculated 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 calculated 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 alter a hashcluster with the SQL command ALTER CLUSTER:

```
ALTER CLUSTER emp_dept . . . ;
```

The implications for altering a hash cluster are identical 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 “Altering Clusters” on page 8-51.

Dropping Hash Clusters

A hash cluster is dropped using the SQL command DROP CLUSTER:

```
DROP CLUSTER emp_dept;
```

A table in a hash cluster is dropped using the SQL command DROP TABLE. The implications of dropping hash clusters and tables in hash clusters are the same for index clusters. See “Dropping Clusters, Clustered Tables, and Cluster Indexes” on page 8-52.

Miscellaneous Management for Schema Objects

The following sections explain miscellaneous topics regarding the management of the various schema objects discussed in this chapter.

Creating Multiple Tables and Views in One Operation

You can create several tables and views and grant privileges in one operation using the SQL command CREATE SCHEMA. The CREATE SCHEMA command is useful if you want to guarantee the creation of several tables and views and grants in one operation; if an individual table or view creation fails or a grant fails, the entire statement is rolled back and none of the objects are created or the privileges granted. For example, 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 command does not support ORACLE extensions to the ANSI CREATE TABLE and CREATE VIEW commands; this includes the STORAGE clause.

Privileges Required to
Create Multiple Schema
Objects

To create schema objects using the CREATE SCHEMA command, 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.

Renaming Schema Objects

You can rename schema objects using two different methods: drop and re-create the object, or rename the object using the SQL command RENAME. If you drop an object and re-create it, all privilege grants for the object are lost. Privileges must be regranted when the object is recreated. Alternatively, a table, view, sequence, or a private synonym of a table, view, or sequence can be renamed using the RENAME command. When using the RENAME command, 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, you should understand the following effects:

- All views and PL/SQL program units dependent on a renamed object become invalid (must be recompiled before next use). See page 8-75 for more information about how ORACLE manages such dependencies.
- All synonyms for a renamed object return an error when used.

Privileges Required to
Rename an Object

To rename an object, you must own the object.

Analyzing Tables, Indexes, and Clusters

You can analyze a table (non-clustered or clustered), index, or cluster to gather data about it or to verify the validity of its storage format.



- A table, index, or cluster can be analyzed to collect or update statistics about the object. When a DML statement is issued, the statistics for the referenced objects are used to determine the most efficient execution plan for the statement; such optimization is called “cost-based optimization.” The statistics are stored in the data dictionary. (For more information about analyzing tables, indexes, and clusters for performance statistics and the optimizer, see the *ORACLE7 Server Application Developer’s Guide*.)
- A table, index, or cluster can be analyzed to *validate* (confirm the integrity of) the structure of the object. For example, in rare cases (due to hardware or other system failures) an index can become corrupted and not perform its job 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. This result is useful in determining whether you have enough room for updates to rows; for example, it can show whether PCTFREE is set appropriately for the table or cluster.

Updating and Deleting
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 command ANALYZE with the STATISTICS option. Such statistics can be used by ORACLE when cost-based optimization is used to choose the most efficient execution plan for SQL statements that access analyzed objects. The database administrator or application developer

can also use statistics generated by this command to write efficient SQL statements that access analyzed objects.

Statistics can be computed or estimated when using the ANALYZE command, as specified by the COMPUTE STATISTICS or ESTIMATE STATISTICS options, respectively:

- When computing statistics, an entire object is scanned to gather data about the object. The data is then used by ORACLE to compute exact statistics about the object. Therefore, slight variances throughout the object are accounted for in 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.
- When estimating statistics, ORACLE gathers representative information from portions of an object. This subset of information is used to provide reasonable, estimated statistics about the object. Therefore, the accuracy of estimated statistics depend on the representativeness of the sampling used by ORACLE. Because only parts of an object are scanned to gather information for estimated statistics, an object can be analyzed quickly. You can optionally specify the number or percentage of rows that ORACLE should use in making the estimate.

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: These tables contain entries (rows) 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.

You can gather the following statistics on a table:

- number of rows
- number of blocks that have been used (computed only)
- 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 (computed only)
- the second largest value per column (computed only)

Note: Statistics for all indexes associated with a table are automatically gathered when the table is analyzed for statistics.

You can gather the following statistics on an index:

- index level (computed only)
- number of leaf blocks
- number of distinct keys
- average number of leaf blocks/key
- average number of data blocks/key
- clustering factor
- minimum key value (computed only)
- maximum key value (computed only)

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;
```

Estimating Statistics for a Schema Object 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 Statistics for a table, index, or cluster can be removed from the data dictionary using the DELETE STATISTICS option of the ANALYZE command. 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. For example, 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 (those 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.

Analyzing Statistics for the Data Dictionary To use cost-based optimization on internal and recursive SQL calls, you have to ensure that ORACLE has gathered statistics on the data dictionary, and that these statistics are up-to-date. If no statistics are gathered on the data dictionary, ORACLE uses rule-based optimization on these calls.

If your site has ORACLE with the procedural option, you can call the following procedures:

DBMS_UTILITY. ANALYZE_SCHEMA()	This procedure takes two arguments, the name of a schema and an analyzation method ('COMPUTE', 'ESTIMATE', or 'DELETE'), and gathers statistics on all of the objects in the schema.
DBMS_DDL. ANALYZE_OBJECT()	This procedure takes four arguments, the type of an object ('CLUSTER', 'TABLE', or 'INDEX'), the schema of the object, the name of the object, and an analyzation method ('COMPUTE', 'ESTIMATE', or 'DELETE'), and gathers statistics on the object.

You should call these procedures periodically to update the statistics.

Whether or not your site has the procedural option, you can alternatively run the script CATSTAT.SQL, which gathers statistics for the objects in the data dictionary. As with the procedures described above, you should run this script periodically to update the statistics. See “Creating Additional Data Dictionary Structures” on page 2-16 for more information.

Validating Tables, Indexes, and Clusters

To verify the integrity of the structure of a table, index, cluster, or snapshot, use the `VALIDATE STRUCTURE` option of the `ANALYZE` command. If the structure is valid, no error is returned. However, if the structure is corrupt, an error message is returned. (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 check if that rectifies the problem; if it does not, drop and re-create the snapshot.)

For example, 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. For example, the following statement validates the EMP table and all associated indexes:

```
ANALYZE TABLE emp VALIDATE STRUCTURE CASCADE;
```

Listing Chained Rows of Tables and Clusters

The chained rows (and migrated rows) of a table or cluster can be easily detected using the `LIST CHAINED ROWS` parameter of the `ANALYZE` command. The results of this command are stored in a specified table that is created explicitly to accept the information returned by this option of the `ANALYZE` command.

To create an appropriate table to accept data returned by an `ANALYZE... LIST CHAINED ROWS` statement, use the `UTLCHAIN.SQL` script that is provided with ORACLE. The `UTLCHAIN.SQL` script creates a table named `CHAINED_ROWS` in the schema of the user that submits the script. The name and location of the `UTLCHAIN.SQL` script are operating system dependent; see your installation or user's guide.



IUG

After a `CHAINED_ROWS` table is created, you can specify it when using the `ANALYZE` command. 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;
```



For more information about reducing the number of chained and migrated rows in a table or cluster, see the *ORACLE7 Server Application Developer's Guide*.

Privileges Required to Analyze a Table, Index, or Cluster

To analyze a table, cluster, or index, you must own the table, cluster, or index or have the ANALYZE ANY system privilege.

Truncating Tables and Clusters

You might want to 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 three options:

- You can delete the rows of a table using the DELETE command. For example, the following statement deletes all rows from the EMP table:

```
DELETE FROM emp;
```
- You can drop a table and then re-create the table. For example, the following statements drop and then recreate the EMP table:

```
DROP TABLE emp;  
CREATE TABLE emp ( . . . );
```
- You can delete all rows of the table using the SQL command TRUNCATE. For example, the following statement truncates the EMP table:

```
TRUNCATE TABLE emp;
```

The first option, the DELETE command, has several characteristics, which may or may not be desired. If many rows are present in a table (or cluster), 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. 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.

The second option, dropping and recreating a table or cluster, also has several characteristics to consider. For example, all associated indexes, integrity constraints, and triggers are also dropped, all objects that depend on the dropped table (or clustered table) are invalidated, and all grants for the dropped table (or clustered table) are dropped.

In contrast, the TRUNCATE command 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 (grants). A TRUNCATE statement also specifies whether or not space currently allocated for the table is returned to the containing tablespace after truncation.

As a TRUNCATE statement deletes rows from a table (or clustered table), triggers associated with the table are not fired. Also, a TRUNCATE statement does not generate any audit information corresponding to DELETE statements if auditing is enabled. Instead, a single audit record is generated for the TRUNCATE statement being issued. (See Chapter 13 for more information about auditing.)

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, the DELETE command can be used or the table must be dropped and re-created.

The REUSE STORAGE or DROP STORAGE options of the TRUNCATE command control whether or not 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 or cluster (or associated indexes) are not changed as a result of the truncation.

Privileges and
Prerequisites Required to
Truncate a Table or Cluster

A user can truncate any table or cluster in the user's associated schema. Also, any user that has the DELETE ANY TABLE system privilege can truncate a table or cluster in any schema.

Before truncating a table (or cluster that contains a table) that contains a parent key, all referencing foreign keys in different tables must be disabled; a self-referential constraint does not have to be disabled.

Enabling and Disabling Triggers

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.
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.

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.

By default, triggers are enabled when first created. You disable a trigger using the ALTER TRIGGER command with the DISABLE option. For example, 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 command with the DISABLE clause and the ALL TRIGGERS option. For example, to disable all triggers defined for the INVENTORY table, enter the following statement:

```
ALTER TABLE inventory  
  DISABLE ALL TRIGGERS;
```

Enabling Triggers

Once you have completed the task that required the trigger to be disabled, you can reenable the trigger so that it fires when appropriate.

You enable a disabled trigger using the ALTER TRIGGER command with the ENABLE option. For example, to enable the disabled trigger named REORDER of the INVENTORY table, enter the following statement:

```
ALTER TRIGGER reorder ENABLE;
```

To enable all triggers defined for a specific table, use the ALTER TABLE command with the ENABLE clause with the ALL TRIGGERS option.

For example, to enable all triggers defined for the INVENTORY table, enter the following statement:

```
ALTER TABLE inventory
  ENABLE ALL TRIGGERS;
```

Privileges Required to
Enable and Disable
Triggers

To enable or disable triggers using the ALTER TABLE command, 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 command, you must own the trigger or have the ALTER ANY TRIGGER system privilege.

Managing Integrity Constraints

This section explains the mechanisms and procedures for manually enabling and disabling integrity constraint, and dropping constraints.

An integrity constraint defined on a table can be in either of two modes:

enabled	When a constraint is enabled, the rule defined by the constraint is enforced on the data values in the columns that define the constraint. The definition of the constraint is stored in the data dictionary.
disabled	When a constraint is disabled, 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.

An integrity constraint can be thought of as a statement about the data in a database. This statement is always not false when the constraint is enabled. However, the statement may or may not be true when the constraint is disabled because data in violation of the integrity constraint can be in the database.

To enforce the rules defined by integrity constraints, the constraints should always be enabled. In certain situations, however, it is desirable to temporarily disable the integrity constraints of a table for performance reasons. For example:

- when loading large amounts of data into a table using SQL*Loader
- 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 turning off integrity constraints can improve the performance of the operation.

While a constraint is enabled, no row that would violate the constraint can be inserted into the table. While the constraint is disabled, though, such a row can be inserted; this row is known as an *exception* to the constraint. While exceptions to a constraint exist in a table, **the constraint cannot be enabled**. The rows that violate the constraint must be either updated or deleted in order for the constraint to be enabled.

You can identify exceptions to a specific integrity constraint while attempting to enable the constraint. See “Reporting Constraint Exceptions” on page 8-73.

Understanding Indexes Associated with Constraints

An index associated with a UNIQUE key or PRIMARY KEY constraint is automatically created by ORACLE when the constraint is enabled, and dropped when the constraint is disabled or dropped; no action is required by the user in either case to manage the index. These associated indexes affect how you manage UNIQUE key and PRIMARY KEY constraints.

When disabling or dropping UNIQUE key and PRIMARY KEY integrity constraints, consider the following issues:

- The constraint’s associated index will be dropped when the constraint is dropped or disabled.
- While enabled foreign keys reference a primary or unique key, you cannot disable or drop the primary or unique key Constraint.

If the constraint is subsequently enabled or redefined, ORACLE creates another index for the constraint.

Because unique and primary keys have associated indexes, you should factor in the cost of dropping and creating indexes when considering whether or not 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.

Enabling and Disabling Integrity Constraints Upon Definition

When an integrity constraint is defined in a CREATE TABLE or ALTER TABLE statement, it can be enabled by including the ENABLE clause in the constraint’s definition or disabled by including the DISABLE clause in the constraint’s definition. If neither the ENABLE nor the DISABLE clause is included in a constraint’s definition, ORACLE automatically enables the constraint.

Enabling Constraints on Definition The following CREATE TABLE and ALTER TABLE statements both define and enable integrity constraints:

```
CREATE TABLE emp (
    empno NUMBER(5) PRIMARY KEY, . . . ;
ALTER TABLE emp
    ADD 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. (See “Reporting Constraint Exceptions” on page 8-73.)

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.

Disabling Constraints on 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 constraints 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.

Enabling and Disabling Existing Integrity Constraints

The ALTER TABLE command can be used to:

- enable a disabled constraint, using the ENABLE clause
- disable an enabled constraint, using the DISABLE clause

Enabling Disabled Constraints The following statements enable disabled integrity constraints:

```
ALTER TABLE dept
    ENABLE CONSTRAINT dname_ukey;
ALTER TABLE dept
    ENABLE PRIMARY KEY,
    ENABLE UNIQUE (dname, loc);
```

An ALTER TABLE statement that 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 is not enabled. (See “Reporting Constraint Exceptions” on page 8-73.)

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.

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);
```

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. Drop the constraint using the ALTER TABLE command with the DROP clause. For example, the following statements both 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 Dropping UNIQUE key and PRIMARY KEY constraints drops the associated 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 no exceptions are present when a CREATE TABLE. . . ENABLE. . . or ALTER TABLE. . . ENABLE. . . statement is issued, the integrity constraint is enabled and all subsequent DML statements are subject to the enabled integrity constraints.

If exceptions exist when a constraint is enabled, an error is returned and the integrity constraint remains disabled. When a statement is not successfully executed because integrity constraint exceptions exist, the

statement is rolled back. If exceptions exist, you cannot enable the constraint until all exceptions to the constraint are either updated or deleted.

To determine which rows violate the integrity constraint, include the EXCEPTIONS option in the ENABLE clause of a CREATE TABLE or ALTER TABLE statement. The EXCEPTIONS option places the ROWID, table owner, table name, and constraint name of all exception rows into a specified table. For example, the following statement attempts to enable 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;
```

Note: You must create an appropriate exceptions report table to accept information from the EXCEPTIONS option of the ENABLE clause before enabling the constraint. An exception table can be created by submitting the script UTLEXCPT.SQL. (The exact name and location of the UTLEXCPT.SQL script are operating system specific; see your installation or user's guide.) The script creates a table named EXCEPTIONS. You can create additional exceptions tables with different names by modifying and resubmitting the script.

If duplicate primary key values exist in the DEPT table and the name of the PRIMARY KEY constraint on DEPT is SYS_C00301, the following rows might be placed in the table EXCEPTIONS by the previous statement:

```
SELECT * FROM exceptions;
```

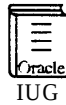
ROWID	OWNER	TABLE_NAME	CONSTRAINT
000003A5.000C.0001	SCOTT	DEPT	SYS_C00301
000003A5.000D.0001	SCOTT	DEPT	SYS_C00301

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. For example:

```
SELECT deptno, dname, loc FROM dept, exceptions
WHERE exceptions.constraint = 'SYS_C00301'
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 that contains the constraint. If updating exceptions, you



must change the value that violates the constraint to a value consistent with the constraint or a null (if allowed). 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.

For example, to correct the exceptions in the previous examples, the following transaction might be issued:

```
UPDATE dept SET deptno = 20 WHERE dname = 'RESEARCH';
DELETE FROM exceptions WHERE constraint = 'SYS_C00301';
COMMIT;
```

The goal, when managing exceptions, 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.

Managing Object Dependencies

The following table shows how objects are affected by changes in other objects on which they depend.

TABLE 8-1
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*
ALTER table [ADD column /MODIFY column] RENAME table, sequence, synonym, view	VALID if there are no errors	INVALID
DROP table, sequence, synonym, view, procedure, function, package	None; the object is dropped	INVALID
CREATE view, procedure**	VALID if there are no errors; INVALID if there are syntax or authorization errors	no change*
CREATE OR REPLACE <i>view or procedure**</i>	VALID if there are no errors; INVALID if there are syntax or authorization errors	INVALID
REVOKE <i>object privilege***</i> ON <i>object</i> TO/FROM <i>user</i>	No change	All objects of <i>user</i> that depend on <i>object</i> are INVALID***
REVOKE <i>object privilege***</i> ON <i>object</i> TO/FROM PUBLIC	No change	All objects in the database that depend on <i>object</i> are INVALID***
REVOKE <i>system privilege****</i> TO/FROM <i>user</i>	No change	All objects of <i>user</i> are INVALID****
REVOKE <i>system privilege****</i> TO/FROM PUBLIC	No change	All objects in the database are INVALID****

* May cause dependent objects to be made INVALID, if object did not exist earlier.

** Stand-alone procedures and functions, packages, and triggers.

*** Only DML object privileges, including SELECT, INSERT, UPDATE, DELETE, and EXECUTE; revalidation does not require recompiling.

**** Only DML system privileges, including SELECT/INSERT/UPDATE/DELETE ANY TABLE, 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 such a case, 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, use the ALTER VIEW command with the COMPILE parameter. For example, the following statement recompiles the view EMP_DEPT contained in your schema:

```
ALTER VIEW emp_dept COMPILE;
```

Privileges Required to Manually Recompile a View To recompile a view manually, the view must be contained in your schema or you must have the ALTER ANY TABLE system privilege.

Manually Recompiling Procedures and Functions

To recompile a stand-alone procedure or function, use the ALTER PROCEDURE/FUNCTION command with the COMPILE parameter. For example, the following statement recompiles the stored procedure UPDATE_SALARY contained in your schema:

```
ALTER PROCEDURE update_salary COMPILE;
```

Privileges Required to Manually Recompile a Procedure or Function To recompile a procedure manually, the procedure must be contained in your schema or you must have the ALTER ANY PROCEDURE system privilege.

Manually Recompiling Packages

To recompile either a package body or both a package specification and body, use the ALTER PACKAGE command with the COMPILE parameter. For example, 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;
```

Privileges Required to Manually Recompile a Package To recompile a package manually, the package must be contained in your schema or you must have the ALTER ANY PROCEDURE system privilege.

Manually Recompiling Triggers

Manually recompile an existing trigger, whether it is currently enabled or disabled, using the ALTER TRIGGER command. For example, to force the compilation of the trigger named REORDER, enter the following statement:

```
ALTER TRIGGER reorder COMPILE;
```


Privileges Required to Manually Recompile a Trigger To recompile a trigger manually, you must own the trigger or have the ALTER ANY TRIGGER system privilege.

Managing Object Name Resolution

Object names referenced in SQL statements can consist of several pieces, separated by periods. ORACLE resolves an object name using the following algorithm:

1. 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 1b.
 - 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 1c.
 - 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 1a, 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 1c, 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 above.

Changing Storage Parameters for the Data Dictionary

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 not be able to acquire additional extents (for example, if a data dictionary table needs 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, by adjusting the values of NEXT or PCTINCREASE for the data dictionary table).



Note: Be very careful when you change 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, recreating, 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
TSS\$	tablespaces defined in the database
FILE\$	files that make up the database; indexed by I_FILE1
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_FILE#_BLOCK#	cluster containing SEG\$ and UET\$; indexed by I_FILE#_BLOCK#
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, you are most likely to need to change are the following segments:

- if the free space in your database is very fragmented, C_TS#
- if you have many indexes or many columns in your tables, C_OBJ#
- if you use integrity constraints heavily, CON\$ and C_COBJ#
- if you have a lot of users defined in your database, C_USER#

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, but ORACLE cannot allocate the extent. The error message ORA-1547 (“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 (see “Displaying Segments That Cannot Allocate Additional Extents” on page 8-84) and consider changing the storage parameters for CON\$ or C_COBJ#.

Displaying Information About Schema Objects

The data dictionary provides many views that provide information about the schema objects described in this chapter. 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

Information about a database's extents is contained in the following data dictionary views:

- USER_EXTENTS
- DBA_EXTENTS
- USER_FREE_SPACE
- DBA_FREE_SPACE

The following sections provide examples of using some of the above views.

Example 1
Displaying Schema
Objects By Type

The following query lists all of the objects owned by the user issuing the query:

```
SELECT object_name, object_type FROM user_objects;
```

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. A segment might not be able to be allocated 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).

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 data files to the tablespace. (See Chapter 7.)
- 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 recreate 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.

MANAGING ROLLBACK SEGMENTS

This chapter explains how to manage rollback segments. The following topics are included:

- guidelines for managing rollback segments
- creating and setting storage parameters for rollback segments
- altering rollback segments
- taking rollback segments on and offline
- assigning a transaction explicitly to a rollback segment
- dropping a rollback segment
- monitoring rollback segments



Trusted

If you are using Trusted ORACLE in DBMS MAC mode, see the *Trusted ORACLE7 Server Administrator's Guide* for additional information on managing rollback segments in that environment.



Parallel

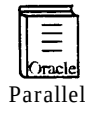
If you are using ORACLE with the Parallel Server option, see the *ORACLE7 Parallel Server Administrator's Guide* for additional information on managing rollback segments in that environment.

Guidelines for Managing Rollback Segments

Here are some guidelines to keep in mind when creating or managing the rollback segments of your databases.

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. If a database is to have other tablespaces, it must have two or more rollback segments in the SYSTEM tablespace. You cannot create any objects in non-SYSTEM tablespaces (not even rollback segments) until you have created and brought online at least one additional rollback segment in the SYSTEM tablespace.
- With the ORACLE Parallel Server, each instance requires access to its own rollback segment (in addition to the SYSTEM rollback segment) in order to start. For additional details, see the *ORACLE7 Parallel Server Administrator's Guide*.
- 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.

Add a Rollback Segment to the SYSTEM Tablespace

An initial rollback segment called SYSTEM is created when a database is created. The SYSTEM rollback segment is created in the SYSTEM tablespace using the default storage parameters associated with that tablespace. You cannot drop this rollback segment.

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. Therefore, after database creation, create at least one additional rollback segment in the SYSTEM tablespace.

Choose Between Public and Private Rollback Segments



Parallel

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 could also have only public segments, as long as the number of segments is high enough that each instance that opens the database can acquire at least one rollback segment in addition to its SYSTEM rollback segment. However, when using the ORACLE Parallel Server, you might want to use private rollback segments. See the *ORACLE7 Parallel Server Administrator's Guide* for more information.

Specify Rollback Segments to Acquire Automatically

By default, when an instance starts, it acquires TRANSACTIONS/TRANSACTIONS_PER_ROLLBACK_SEGMENT rollback segments. If you want to ensure that the instance acquires particular rollback segments (of particular sizes or in particular tablespaces, for example, or specific to an instance in the case of the Parallel Server), specify the rollback segments by name in the parameter ROLLBACK_SEGMENTS 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 (such as batch jobs) perform better with larger rollback segments. In most cases, rollback segments can handle transactions of any size easily; however, in extreme cases when a transactions 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 such problems are avoided.

In contrast, long running transactions work better with larger rollback segments because the rollback entries for a long running transaction can fit in pre-allocated extents of a large rollback segment.

When a database system's 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; dynamic extent allocation and truncation for rollback segments can be minimized. (This is not required for most systems and is intended for extremely large or small transactions.) To optimize for such situations, perform the following steps:

1. 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. Set OPTIMAL for each such rollback segment so that the rollback segment returns to its intended size if it has to grow.
2. Tell the application developers or users about the different sets of rollback segments that correspond to the different types of transactions. In many cases, 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 Extents of Equal Size

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 . Note that PCTINCREASE cannot be specified for rollback segments and defaults to 0. Also note that if the sizes 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

The OPTIMAL parameter for each rollback segment should be carefully chosen according to the kind of transactions that the system runs. 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. For a system that executes long queries on active data, OPTIMAL should also be large in order to avoid "snapshot too old" errors. OPTIMAL should be smaller for a system that mainly executes short transactions and short queries so that the rollback segments remain small enough to be cached in memory, thus improving system performance.

To obtain estimates and to monitor the effectiveness of the OPTIMAL settings for rollback segments, use the MONITOR ROLLBACK feature of SQL*DBA. In this monitor, the following statistics are given for each rollback segment:

<i>Sizes, High Water</i>	the most space ever allocated for the rollback segment, in bytes
<i>Sizes, Optimal</i>	the OPTIMAL size of the rollback segment, in bytes
<i>Occurrences, Wraps</i>	the cumulative number of times a transaction continues writing from one extent in a rollback segment to another existing extent

<i>Occurrences, Extends</i>	the cumulative number of times a new extent is allocated for a rollback segment
<i>Shrinks</i>	the cumulative number of times ORACLE has truncated extents from the rollback segment
<i>Average Sizes, Shrunk</i>	the average size of the space ORACLE truncated from the rollback segment, in bytes
<i>Average Sizes, Active</i>	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 9-1 provides additional information on how to interpret the statistics given in this monitor.

TABLE 9-1
Analyzing the Effectiveness of
Current OPTIMAL Settings

<i>Shrinks</i>	<i>Average Sizes, Shrunk</i>	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 <i>Shrinks</i> is low.

Set the Storage Location for Rollback Segments

If possible, create one tablespace specifically to hold all rollback segments, other than the two required in the SYSTEM tablespace. This way, all rollback segment data is stored separately from other types of data. This rule will 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. (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 over the database's rollback segments.

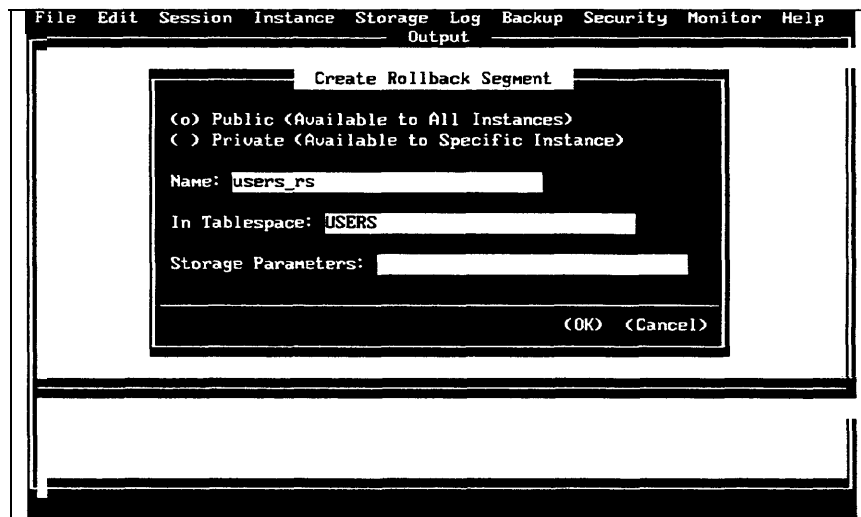
- A tablespace's free extents are likely to be more fragmented if the tablespace contains rollback segments that frequently allocate and deallocate extents.

Creating Rollback Segments

To create additional rollback segments for a database, use either the Create Rollback Segment dialog box of SQL*DBA or the SQL command `CREATE ROLLBACK SEGMENT`. The tablespace to contain the new rollback segment must be online.

The Create Rollback Segment dialog box allows you to create a new rollback segment. Figure 9-1 shows the Create Rollback Segment dialog box.

FIGURE 9-1
The Create Rollback
Segment Dialog



This example creates a public rollback segment named `USERS_RS` in the `USERS` tablespace, using the default storage parameters of the `USERS` tablespace.

The following statement is the command equivalent of the Create Rollback Segment dialog box in Figure 9-1:

```
CREATE PUBLIC ROLLBACK SEGMENT users_rs
TABLESPACE users;
```


Privileges Required to Create Rollback Segments

To create rollback segments, you must have the CREATE ROLLBACK SEGMENT system privilege.

Bringing New Rollback Segments Online

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 9-10 for more information.

If you create a private rollback segment, you should add the name of the new rollback segment to the ROLLBACK_SEGMENTS parameter in the parameter file for the database. Doing so causes the private rollback segment to be captured by the instance at instance startup. 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 example:

```
ROLLBACK_SEGMENTS = (RS1, RS2)
```

Specifying Storage Parameters for Rollback Segments

When you create a rollback segment, you can optionally specify storage parameters for the segment's extents. You can also change storage parameters for an existing rollback segment.

Setting Storage Parameters During Rollback Segment Creation

As an example, 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);

```

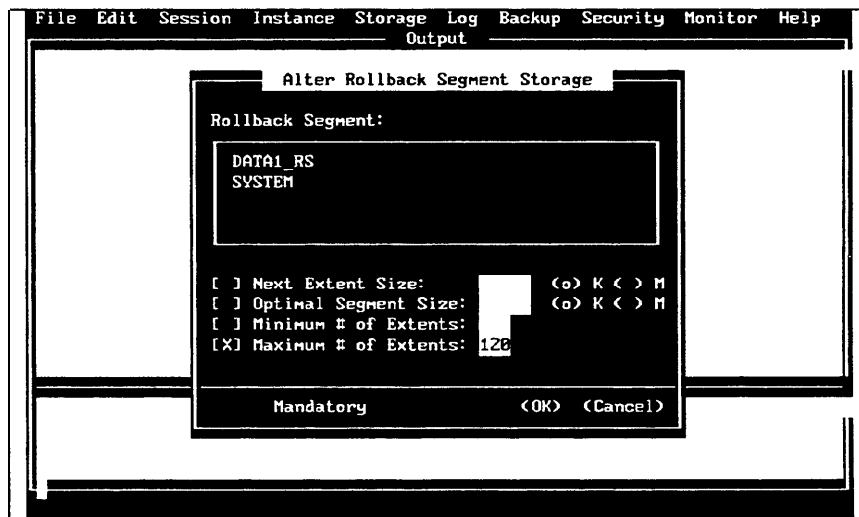
You can also use SQL*DBA's Create Rollback Segment dialog to set the rollback segment's storage parameters.

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 either the Alter Rollback Segment Storage dialog box of SQL*DBA or the SQL command ALTER ROLLBACK SEGMENT. Figure 9-2 shows the Alter Rollback Segment Storage dialog box that alters the maximum number of extents that the DATA1_RS rollback segment can allocate.

FIGURE 9-2
The Alter Rollback Segment Storage Dialog



The following statement is the command equivalent of the Alter Rollback Segment Storage dialog box in Figure 9-2:

```

ALTER PUBLIC ROLLBACK SEGMENT data1_rs
  STORAGE (MAXEXTENTS 120);

```

Note: You can alter the settings for the SYSTEM rollback segment, including the OPTIMAL parameter, just as you can alter those of any rollback segment.

For guidance on setting sizes and storage parameters (including OPTIMAL) for rollback segments, see “Guidelines for Managing Rollback segments” on page 9-2.

Note: If you are altering a public rollback segment, you must include the keyword PUBLIC in the ALTER ROLLBACK SEGMENT command.

Privileges Required to
Alter Rollback Segments

To alter rollback segments, you must have the ALTER ROLLBACK SEGMENT system privilege.

Taking Rollback Segments Online and Offline

A rollback segment is either *online* (available to transactions) or *offline* (unavailable to transactions). In most cases, rollback segments are online and available for use by transactions.

You might want to take online rollback segments offline in the following situations:

- 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 prior to 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 prior to dropping it.

Note: You cannot take the SYSTEM rollback segment offline.

You might later want to bring a rollback segment back online so that transactions can use it. Also, when a rollback segment is created, it is initially offline; you must explicitly bring a newly created rollback segment online before it can be used by an instance’s transactions. An offline rollback segment can be brought online by any instance accessing the database that contains the rollback segment.

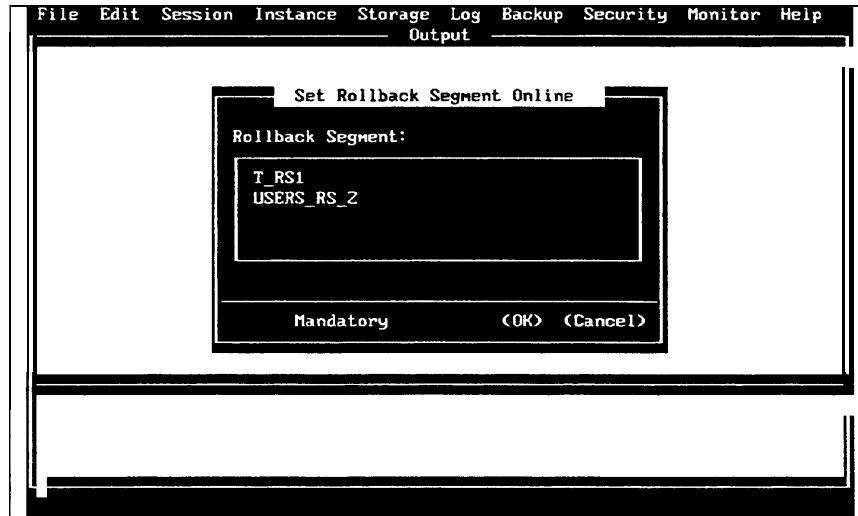
Bringing Rollback Segments Online

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 either the Set Rollback Segment Online dialog box of SQL*DBA or the SQL command ALTER ROLLBACK SEGMENT with the ONLINE option.

Figure 9-3 shows the Set Rollback Segment Online dialog box that brings the rollback segment USER_RS_2 online.

FIGURE 9-3
The Set Rollback Segment
Online Dialog



The following statement is the command equivalent of the Set Rollback segment Online dialog box in Figure 9-3:

```
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 “Displaying Rollback Segment Information” on page 9-16 for a query for checking rollback segment state, and Appendix B for information about DBA_ROLLBACK_SEGS.

Bringing a PARTLY AVAILABLE Rollback Segment Online A rollback segment in the PARTLY AVAILABLE state (its status in the data dictionary view DBA_ROLLBACK_SEGS is “PARTLY AVAILABLE”) contains data for an in-doubt distributed transaction. 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.

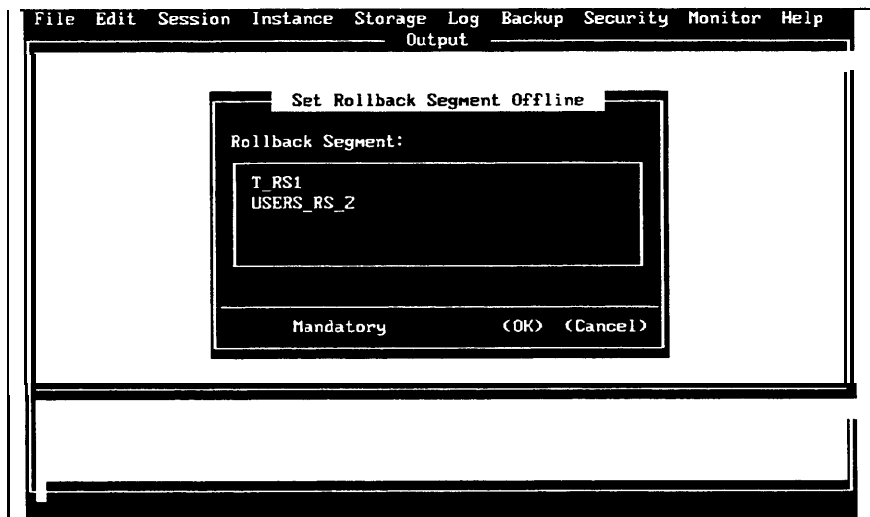
Bringing a Rollback Segment Online Automatically If you would like a rollback segment to be automatically brought online whenever you start up the database, add the segment's name to the ROLLBACK_SEGMENTS parameter in the database's parameter file. (See Appendix A for information.)

Taking Rollback Segments Offline

To take an online rollback segment offline, use either the Set Rollback Segment Offline dialog box of SQL*DBA or 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.

Figure 9-4 shows the Set Rollback Segment Offline dialog box that takes the rollback segment USER_RS_2 offline.

FIGURE 9-4
The Set Rollback Segment Offline Dialog



The following statement is the command equivalent of the Set Rollback Segment Offline dialog box in Figure 9-4:

```
ALTER ROLLBACK SEGMENT user_rs_2 OFFLINE;
```

Checking Whether a Rollback Segment Went Offline

If you try to take offline a rollback segment that does not contain active rollback entries, ORACLE immediately takes the segment offline and changes its status to “OFFLINE”.

In contrast, if you try to take offline a rollback segment that does contain rollback data for active transactions (whether local, remote, or distributed), ORACLE makes the rollback segment unavailable to future transactions and takes it offline once 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. (See “Displaying Rollback Segment Information” on page 9-16 for information on viewing rollback segment status; also see Appendix B for information about the views DBA_ROLLBACK_SEGS and V\$ROLLSTAT.)

The instance that tried to take a rollback segment offline and caused it to be 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* rollback segment offline, it remains offline (even if the instance is shut down and restarted) until you bring it back online. After you take a *private* rollback segment offline, it remains offline until you explicitly bring it back online *or* you restart the instance (unless the private rollback segment’s name is removed from the list in the parameter file).

Privileges Required to Take Rollback Segments Online and Offline

To take a rollback segment online or offline, you must have the ALTER ROLLBACK SEGMENT system privilege.

Explicitly Assigning a Transaction to a Rollback Segment

A transaction can be explicitly assigned to a specific rollback segment using the SET TRANSACTION command with the USE ROLLBACK SEGMENT parameter. Transactions are explicitly assigned to rollback segments so that:

- 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 statement 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, the next transaction will be automatically assigned to any available rollback segment by ORACLE unless the new transaction is explicitly assigned to a specific rollback segment by the user.

Privileges Required to Assign a Transaction to a Rollback Segment

No special privileges are required to assign a transaction to a specific rollback segment explicitly; any user can do this.

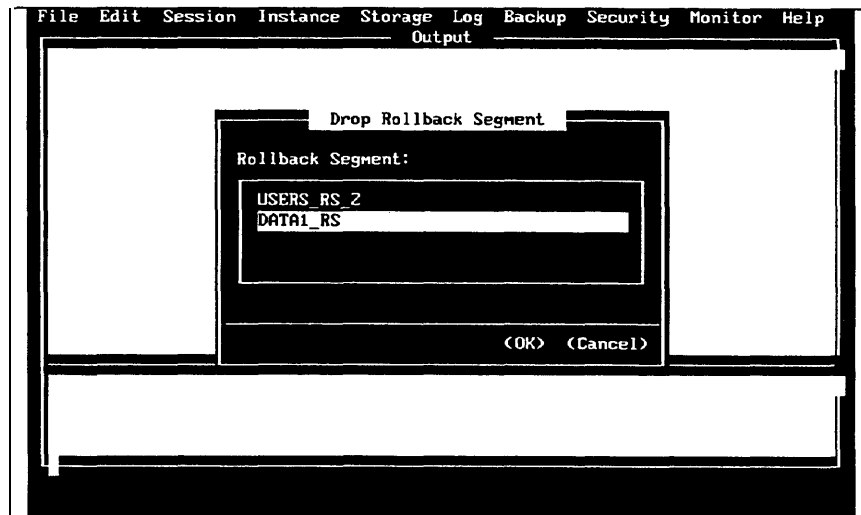
Dropping Rollback Segments

Rollback segments can be dropped 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; see “Taking Rollback Segments Offline” on page 9-12.

If a rollback segment is offline, you can drop it using either the Drop Rollback Segment dialog box of SQL*DBA or the SQL command DROP ROLLBACK SEGMENT. Figure 9-5 shows the Drop Rollback Segment dialog box that drops the DATA1_RS rollback segment.

FIGURE 9-5
The Drop Rollback
Segment Dialog



The following statement is the command equivalent of the Drop Rollback Segment dialog box in Figure 9-5:

```
DROP PUBLIC ROLLBACK SEGMENT data1_rs;
```

If you use the DROP ROLLBACK SEGMENT command, 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.

Once 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 “Displaying Rollback Segment Information” on page 9-16 and Appendix B for more information about the view DBA_ROLLBACK_SEGS.)

Privileges Required to Drop Rollback Segments

To drop a rollback segment, you must have the DROP ROLLBACK SEGMENT system privilege.

Monitoring Rollback Segment Information



Use the MONITOR ROLLBACK feature of SQL*DBA to monitor a rollback segment’s size, number of extents, optimal number of extents, activity concerning dynamic deallocation of extents, and current usage by active transactions; this feature is explained in the *ORACLE7 Server Utilities User’s Guide*. Also see “Set an Optimal Number of Extents for Each Rollback Segment” on page 9-5 for a detailed description of how to use the MONITOR for the corresponding operation.

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, table space_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

Example 1
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

Example 2
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.

Example 3
Displaying Public and
Private 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

Example 4
Displaying 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

ESTABLISHING SECURITY POLICIES

This chapter provides guidelines for developing security policies for the following areas of database operation:

- determining who manages and monitors the security of each database
- setting the level of system security
- setting the level of object security
- setting the level of security for the different types of users:
 - end-user security
 - administrator security
 - application developer security
- establishing a database auditing policy



Trusted

If you are using Trusted ORACLE, you must consider additional security issues; see the *Trusted ORACLE7 Server Administrator's Guide*.

The Security Administrator

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.

Once it has been decided 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.

System Security Policy

The overall security of a database system should include many considerations. This section discusses issues relevant to any database system.

Management of Database Users

Users (or, more precisely, 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 or there may be a number of administrators with the 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 operating system or using the database. In most cases, user authentication by the host operating system is preferred for the following reasons:

- Users can connect to ORACLE faster and more conveniently (without specifying a user name or password).
- Centralized control over user authorization in the operating system: ORACLE need not store or manage user passwords and user names if the operating system and database correspond.

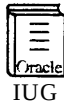
- User entries in the database and operating system audit trails correspond.

User authentication by the database is normally used when the host operating system cannot support user authentication. See “Creating Users” on page 11-7 for more information about user authentication.

Operating System Security

If applicable, certain security considerations must also be made for the operating system environment that executes ORACLE and any database applications. For example:

- 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 is used to identify database roles for users, the security administrators must have the operating system privileges to modify the security domain of operating system accounts.



See our installation or user's guide for more information concerning operating system security issues for ORACLE databases.

Data Security Policy

Decisions must be made regarding the security policy of the data in a database. For example, it may be acceptable to have very little data security in a database; it may be tolerable to allow any user to create any schema object and grant access privileges for their objects to any other user of the system. Alternatively, it might be necessary that data security be very controlled; a database or security administrator might be 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 lax. However, if data is sensitive, a discreet security policy should be developed to maintain tight control over access to objects.

User Security Policy

Decisions must be made regarding the security policy of the users of a database. Specific issues must be considered for each type of database user, including end-users, administrators, and application developers.

General User Security

For all types of database users, consider the following issues:

Password Security

If user authentication is managed by the database, a password security policy should be developed to maintain database access security. For example, database users should be required to change their passwords at regular intervals or if their passwords are revealed to another person. By forcing a user to modify passwords in such situations, unauthorized database access can be reduced if users mistakenly reveal their passwords.

Privilege Management

General decisions must be made for privilege management for all types of users. For example, in a database with a large number of user names, it may be beneficial to use roles to manage the privileges available to users. Alternatively, in a database with a handful of user names, it may be easier to grant privileges explicitly to users and avoid the use of roles.

A database that has a large number of 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

A policy must be made regarding end-user security. For example, if the 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.

Example

Consider the following situation:

- Every user in the accounting department of a company needs the privileges to run the ACCTS_RECEIVABLE and ACCTS_PAYABLE database applications.

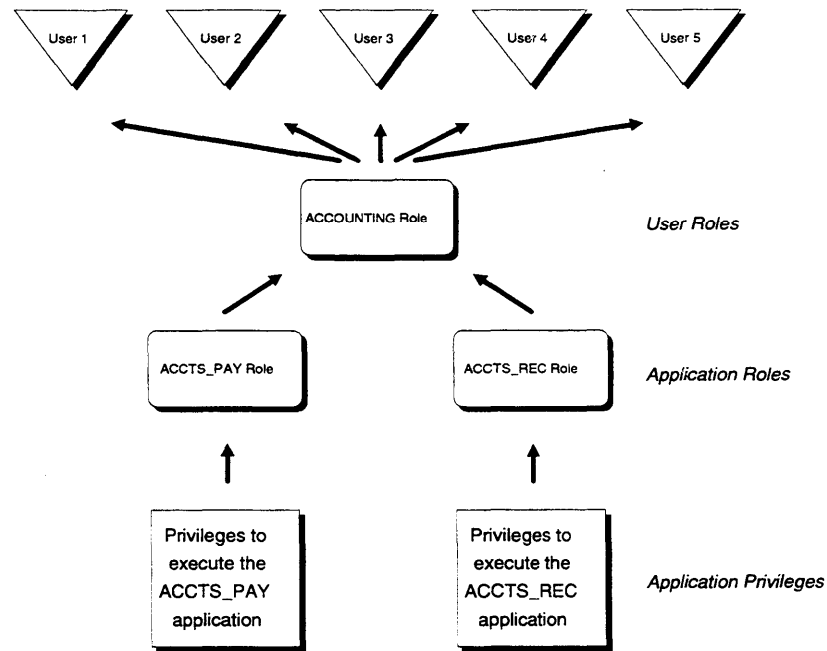
- Roles are associated with both applications, and contain the object privileges necessary to execute those applications.

To address this simple security situation, the following actions should be performed by the database administrator or security administrator:

1. A role named ACCOUNTANT should be created.
2. The roles for the ACCTS_RECEIVABLE and ACCTS_PAYABLE database applications should be granted to the ACCOUNTANT role.
3. The ACCOUNTANT role should be granted to each user of the accounting department.

This security model is illustrated in Figure 10-1.

FIGURE 10-1
User Roles



This plan addresses the following potential future 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.

Roles should be utilized in all possible situations to make end-user privilege management efficient and simple.

Administrator Security

A policy must be made regarding administrator security. For example, if 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, if the database is small and has only a few administrators, it maybe 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 user names to prevent unauthorized access to the database. Connections as SYS and SYSTEM allow a user the powerful privileges to modify a database in many ways. Therefore, use of these user names are extremely sensitive, and should only be available to select database administrators.

The passwords for these accounts can be modified using the procedures described in "Altering Users" on page 11-11.

Protection for Connections as INTERNAL

Only database administrators should have the capability to connect to a database using the keyword INTERNAL. Connections as INTERNAL allow 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). Therefore, use of this keyword is extremely privileged, and should only be available to select database administrators.

If your ORACLE Server supports role identification using the host operating system, the INTERNAL keyword is protected by the OSOPER and OSDBA roles. Each role provides a different level of

functionality when connected as INTERNAL. See “Alternatives to INTERNAL: OSOPER and OSDBA” on page 1-5 for more information about the OSOPER and OSDBA roles and their functions.

Note: To be able to configure a database administrator’s operating system account with these roles, the security administrator must have the ability to modify the security for operating system accounts.

If your ORACLE Server does not support role identification via the host operating system, the use of INTERNAL is protected in an operating system specific method (usually by passwords). See “Connecting as INTERNAL” on page 1-4 and your installation or user’s guide for details.



Using Roles for Administrator Privilege Management

Roles are the easiest way to restrict the powerful system privileges and roles required by administrative personnel of the database.

Example

Assume that the database administrator responsibilities at a large installation are shared among several database administrators, each responsible for specific database management jobs. There might be:

- an administrator responsible for object creation and maintenance
- an administrator responsible for database tuning and performance
- a security administrator, who is responsible for creating new users, granting roles and privileges to database users, etc.
- 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 who need limited capabilities to experiment with database management

In this situation, 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 will easily solve the following potential future problems:

- 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

A special policy must be made for the application developers using a database. For example, the privileges to create necessary objects can be granted to application developers. Alternatively, the privileges to create objects can only be granted to a database administrator, who receives requests for object creation from developers.

Application Developers and Their Privileges

The database application developer is a unique type of database user who requires special groups of privileges to accomplish his job. 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. By imposing this restriction:

- application developers do not compete with end-users for database resources
- application developers cannot detrimentally affect a production database

When an application has been thoroughly developed and tested, the application is allowed to access the production database and made available to the appropriate end-users of the production database.

Free vs. Controlled Application Development

The database administrator has two options when determining the privileges that 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 soon 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 could be purely one of the two options, other systems could be a mix of the options. 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 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

Roles can be created by the security administrator 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. Notice two important points 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 domain), such as CREATE ANY TABLE, CREATE ANY PROCEDURE, and so on, 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 normally 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 individual application developer:

- the tablespaces in which the developer can create tables or indexes
- the quota for each tablespace accessible to the developer

Both limitations can be set by altering a developer's security domain. For more information, see "Altering Users" on page 11-11.

Application Administrator Security

In large database systems with many database applications (for example, precompiled applications and SQL*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

In many cases, an application administrator is also the application developer that 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.

Auditing Policy

A policy should be developed for the auditing procedures of each database. For example, it may be decided 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, more general system auditing is followed by more specific types of auditing once the origins of suspicious activity are determined.

11

MANAGING USERS AND RESOURCES

This chapter explains how to control access to an ORACLE database as a whole. Topics include:

- complying with your ORACLE license agreement
- creating, managing, and dropping database users
- creating, assigning, altering, and dropping profiles (resource limits)
- enabling and disabling resource limits
- displaying information about users and profiles

For guidelines on establishing security policies for users and profiles, see Chapter 10.

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 12.



Trusted

For databases with Trusted ORACLE, see the *Trusted ORACLE7 Server Administrator's Guide* for additional information about user management in that environment.

Managing 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 user name defined in the database. This section explains how to create, manage, and drop users for a database.

Using Session and User Licensing

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 instead 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 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 (concurrent usage licensing or named user licensing), but not both.

Note: In a few cases, your site might not use either of these types of licensing. For example, your site might have an unlimited license. In these cases only, leave the licensing mechanism disabled: omit the parameters `LICENSE_MAX_SESSIONS`, `LICENSE_SESSIONS_WARNING`, and `LICENSE_MAX_USERS` from the parameter file, or set all three to 0.

The following sections explain how to set the concurrent usage or named user limit, and how to view current settings and the highest number of concurrent sessions.

Using Concurrent Usage Licensing

Concurrent usage licensing limits the number of sessions that can be connected to the database on the specified computer at a time. 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; see “Edit New Parameter Files” on page 2-5. You can also change the maximum number of concurrent sessions while the database is running.

Once this limit is reached, only users with `RESTRICTED SESSION` privilege (usually DBAs) can connect to the database; when such a user connects, ORACLE sends him a message saying 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 kill unneeded processes. (See “Killing Sessions” on page 4-13.) Do not raise the

licensing limits unless you have upgraded your ORACLE license agreement.

In addition to setting a maximum, 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 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 as INTERNAL, 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” to determine if you need to upgrade the ORACLE license for your site; see “Viewing Licensing Limits and Current Values” on page 11-5.



Parallel

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. See the *ORACLE7 Parallel Server Administrator's Guide* for more information about setting and changing these limits in that environment.



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 take that into consideration and set the maximum concurrent usage limit lower to account for the multiplexed sessions.

The following sections explain how to set and enforce the concurrent usage limit.

Setting the Maximum Number of Sessions To set the maximum number of concurrent sessions for an instance, set the parameter LICENSE_MAX_SESSIONS. For example:

```
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, since 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 Runs 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. For example, the following statement changes the maximum limit to 100 concurrent sessions:

```
ALTER SYSTEM SET LICENSE_MAX_SESSIONS = 100;
```

As another example, 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 runs, you must have the `ALTER SYSTEM` privilege. Also, to connect to an instance once the instance's maximum limit has been reached, you must have the `Restricted SESSION` privilege.



Do not raise the concurrent usage limits unless you have appropriately upgraded your ORACLE Server license; contact your Oracle representative for information.

Using Named User Limits Named user licensing limits the number of individuals who are 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. Once this limit is reached, you cannot create more users; if you try to do so, ORACLE returns an error saying that the maximum number of users have been created, and writes a message to the `ALERT` file.

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 connect using the same user name.

For instances running with the Parallel Server, all instances connected to the same database should have the same named user limit. See the *ORACLE7 Parallel Server Administrator's Guide* for more information on setting this limit in that environment,



Parallel

The following sections explain how to set and use the named user limit.

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. For example:

```
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, and you should either 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. For example, 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.



Do not raise the named user limit unless you have appropriately upgraded your ORACLE license; contact your Oracle representative for 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.

For example:

```
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	88	250

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
```

Authenticating Users

A user connecting to an instance can be authenticated by either the operating system or the database. For simplicity, you might want all users to be authenticated in the same way, but each user can be authenticated a different way; some users can be authenticated by the operating system, while others are authenticated by ORACLE. You specify individually for each user how that user is authenticated when you create the user, and you can later alter any user to change her method of authentication.

Using Operating System Authentication



If your operating system permits (see your installation or user's guide), you can have it authenticate users. If you do so, set the parameter `OS_AUTHENT_PREFIX`, and use this prefix in ORACLE user names. This parameter defines a prefix that ORACLE adds to the beginning of every user's operating system account name. ORACLE compares the prefixed user name with the ORACLE user names in the database when a user attempts to connect.

For example, assume that `OS_AUTHENT_PREFIX` is set as follows:

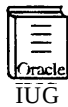
```
OS_AUTHENT_PREFIX = OPS$
```

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, and therefore look like “OPS\$TSMITH”.

The default value of this parameter is “OP\$” 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 user names exactly match operating system user names.

Once you set OS_AUTHENT_PREFIX, it should remain the same for the life of a database. If you change the prefix, any database user name that includes the old prefix cannot be used to establish a connection, unless you alter the user name to have it use password authentication.



Note: The text of the OS_AUTHENT_PREFIX parameter is case-sensitive on some operating systems. See your installation or user's guide for more information about this initialization parameter.

Operating System Authentication and the Multi-Threaded Server If you want to have the operating system authenticate a user, by default that user 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 network connection.

If you are not concerned with this security risk and want to use operating system user authentication with the multi-threaded server, set the parameter REMOTE_OS_AUTHENT 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.)

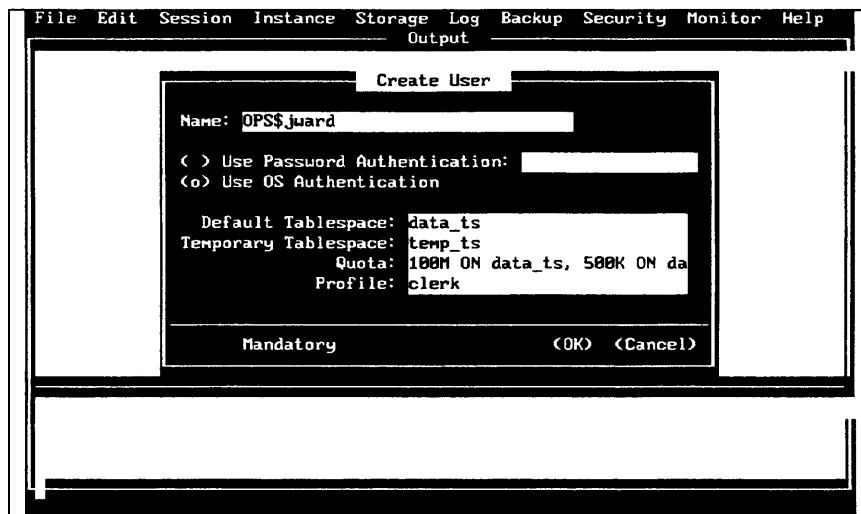
Using Database Authentication

To have ORACLE authenticate a user, specify a password for the user when you create or alter the user. Any user can change his 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.

Creating Users

You create a user with either the Create User dialog box of SQL*DBA or the SQL command CREATE USER. Using either option, you can also specify the new user's default and temporary segment tablespaces, tablespace quotas, and profile. Figure 11-1 shows the Create User dialog box.

FIGURE 11-1
The Create User Dialog



The following statement is the command equivalent of the Create User dialog box in Figure 11-1:

```
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;
```

Note: A newly-created user cannot connect to the database until he is granted the CONNECT system privilege. See “Granting and Revoking Privileges and Roles” on page 12-15.

Specifying a Name

Within each database, a username must be unique with respect to other user names 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 unique names. See Chapter 8 for more information about schema objects and their naming rules, and Chapter 12 about roles and their naming rules.

User Names in Multi-Byte Character Sets In a database that uses a multi-byte character set, Oracle Corporation strongly recommends that each user name contain at least one single-byte character. If a user name contains only multi-byte characters, the encrypted user name/password combination is considerably less secure.

Setting a User's Authentication

In the previous example, the new user is to be authenticated using the operating system. The user name 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 user name accordingly, by omitting the prefix or substituting the correct prefix.

Alternatively, you can create a user so that he is authenticated using the database and a password; the following statement is an example:

```
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.

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 will not create any type of object, this default setting is fine. However, if a user can create any type of object, consider specifically setting the user's default tablespace. You can set 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. See "Assigning Tablespace Quotas" on page 11-10.
- 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 data files.

In the example above, 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 example above, JWARD's temporary tablespace is TEMP_TS, a tablespace created explicitly to contain temporary segments only.

Assigning Tablespace Quotas

You can assign each user a tablespace quota for any tablespace. Assigning a quota does two things:

- If the user has a privilege to create some type of object, the user can create 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 some type of schema object, you must assign a quota to allow the user to create objects; at minimum, assign the user a quota for his default tablespace, and assign additional quotas for other tablespace in which the user 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 altering the user's current quota to zero. Once a quota of zero

is assigned, the user's objects in the revoked tablespace remain, but the objects cannot be allocated any new space.

The 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 advantages and disadvantages 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; see "Altering Users" on page 11-11.

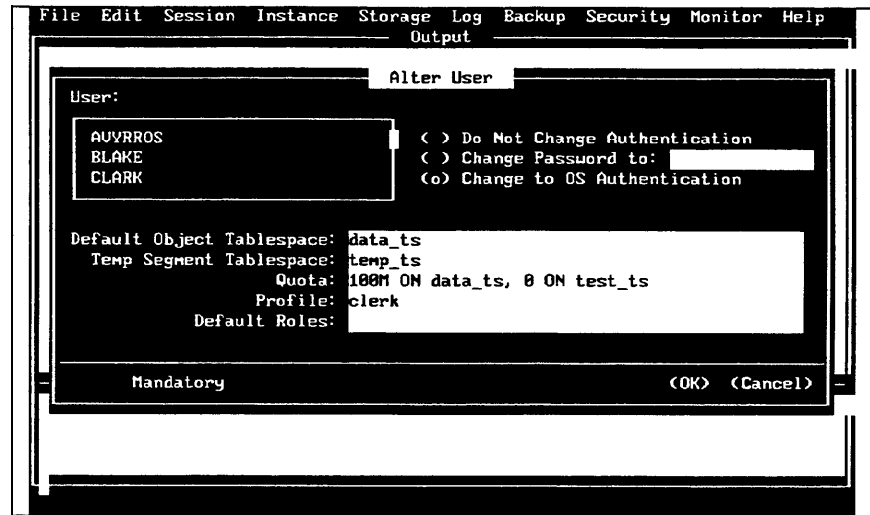
Privileges Required to Create Users

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. Due to such privileged power, a security administrator is normally the only type of user that has the CREATE USER system privilege.

Altering Users

You can alter a user's security settings with either the Alter User dialog box of SQL*DBA or the SQL command ALTER USER. Changing a user's security settings affects the user's future sessions, not current sessions. Figure 11-2 shows the Alter User dialog box.

FIGURE 11-2
The Alter User Dialog



The following statement is the command equivalent of the Alter User dialog box in Figure 11-2:

```
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 above examples change AVYRROS's security setting as follows:

- Authentications 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 Password, for Non-DBAs

Most non-DBA users do not use SQL*DBA, but can still change their own passwords with the ALTER USER command, as in the following example:

```
ALTER USER andy
  IDENTIFIED BY swordfish;
```

Any user can change her own password this way, without any special privileges (other than those to connect to the database). Users should be encouraged to change their passwords themselves frequently.

A user must have the ALTER USER privilege to change between ORACLE authorization and operating system authorization; 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.

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. Any role directly granted to a user can potentially be a default role of the user; an indirectly granted role (a role that is granted to a role) cannot be a default role. The number of default roles for a user should not exceed the maximum number of enabled roles that are allowed per user or, when the user tries to connect, errors are returned and the connection is not allowed.

Note: ORACLE automatically enables a user's default roles when the user creates a session. Placing a role in a user's list of default roles bypasses authentication for the role, whether the role is defined to be authorized using a password or the operating system.

If you specify a list of roles, all other roles granted to that user are removed from the user's default role list. For example, suppose user AVYRROS has been granted the roles DEVELOPER and CLERK, and CLERK is his only default role. The following statement removes CLERK from his default role list and adds DEVELOPER.

```
ALTER USER avyrros
  DEFAULT ROLE DEVELOPER;
```

In this case, any roles subsequently granted to AVYRROS will not be default roles, and will be disabled on connection.

If you specify ALL for the user's list of default roles, every role granted directly to the user is automatically added to the user's list of default roles. Subsequent modification of a user's default role list can remove newly granted roles from a user's list of default roles. The following example causes all roles currently granted to AVYRROS to be added to his list of default roles, as well as all roles granted in the future:

```
ALTER USER avyrros
  DEFAULT ROLE ALL;
```

Furthermore, you can specify ALL EXCEPT with a list of role, and those roles will be the only roles granted to the user not on his default role list. For example, the following statement adds all roles currently granted to AVYRROS except the role PAYROLL to the user's default role list; any roles granted to AVYRROS in the future are also added to the default role list:

```
ALTER USER avyrros  
  DEFAULT ROLE ALL EXCEPT payroll;
```

To cause a user to have no default roles, specify NONE for the user's list of default roles, as in the following example;

```
ALTER USER avyrros  
  DEFAULT ROLE NONE;
```

Changing a user's default role list affects subsequent sessions; it does not affect any session in progress at the time.

Revoking a role from a user automatically removes the role from the user's default role list.

Privileges Required to Alter Users

As noted above, any user can change his own password. 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.

Dropping Users

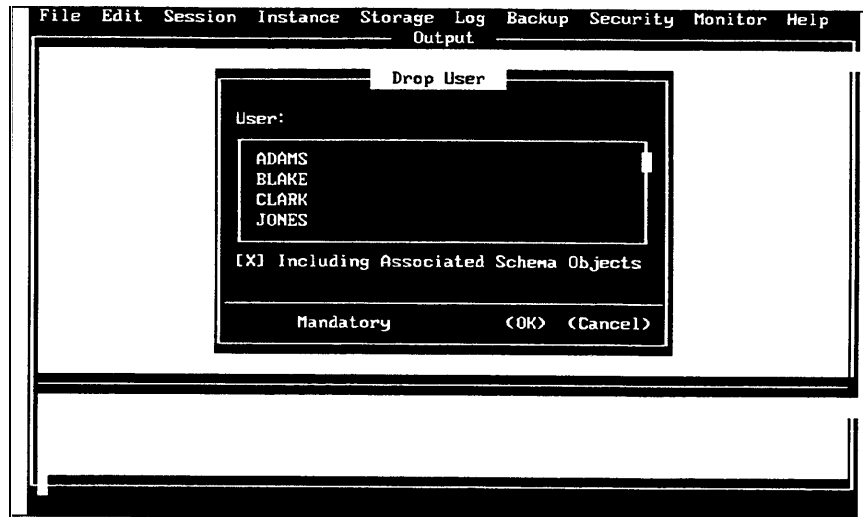
When a user is dropped, the user and associated schema is removed from the data dictionary and all schema objects contained in the user's schema, if any, are immediately dropped.

Note: If a user's schema and associated objects must remain but the user must be revoked 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 kill the user's sessions using either the Kill Session dialog box of SQL*DBA or the SQL command ALTER SYSTEM with the KILL SESSION clause; see "Killing Sessions" on page 4-13.

Drop a user from a database using either the Drop User dialog box of SQL*DBA or the SQL command DROP USER. Figure 11-3 shows the Drop User dialog box.

FIGURE 11-3
The Drop User Dialog



[f the user's schema contains any schema objects, the Including Associated Schema Objects check box must be selected to drop the user and all associated objects and foreign keys that depend on the tables of the user successfully; if you do not select this check box 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 that owns a table, check whether any views or procedures depend on the table.

The following statement is the command equivalent of the Drop User dialog box in Figure 11-3:

```
DROP USER jones CASCADE;
```

Privileges Required to Drop Users

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 normally the only type of user that has this privilege.

Managing Resources with Profiles

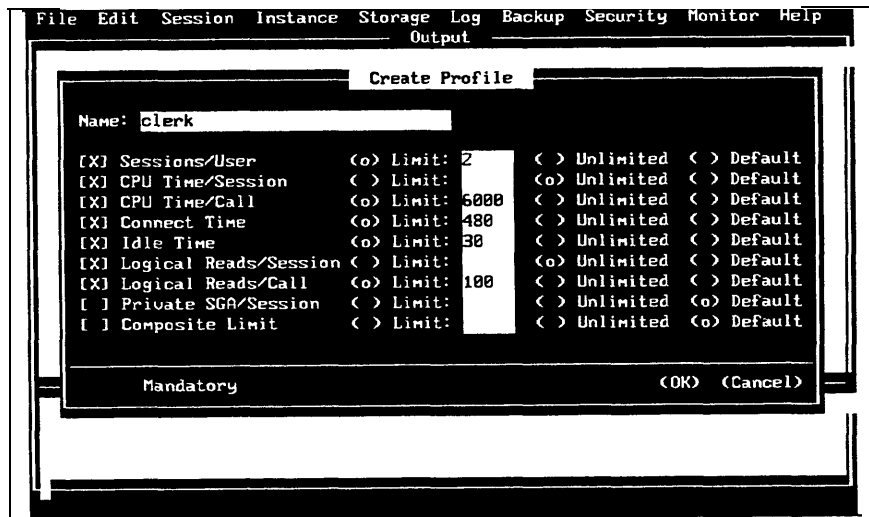
A profile is a named set of resource limits. If resource limits are turned on, ORACLE limits a user's use of database and instance resources to that given in his profile. You can assign a profile to each user, and a default profile to all users who do not have specific profiles. For profiles to take effect, resource limits must be turned on for the database as a whole.

This section explains how to create, assign, alter, and drop profiles, and how to turn resource management on and off.

Creating Profiles

Create profiles using either the Create Profile dialog box of SQL*DBA or the SQL command CREATE PROFILE. At the same time, you can explicitly set particular resource limits. Figure 11-4 shows the Create Profile dialog box.

FIGURE 11-4
The Create Profile Dialog



The following statement is the command equivalent of the Create Profile dialog box in Figure 11-4:

```
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 named DEFAULT. You can also specify limits for the DEFAULT profile. (See “Using the Default Profile” on page 11-17.)

Privileges Required to Create Profiles

To create a profile, you must have the CREATE PROFILE system privilege.

Using the Default Profile

Each database automatically has a default profile named DEFAULT. The default profile’s 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 dialog box or a typical ALTER PROFILE statement, as in:

```
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

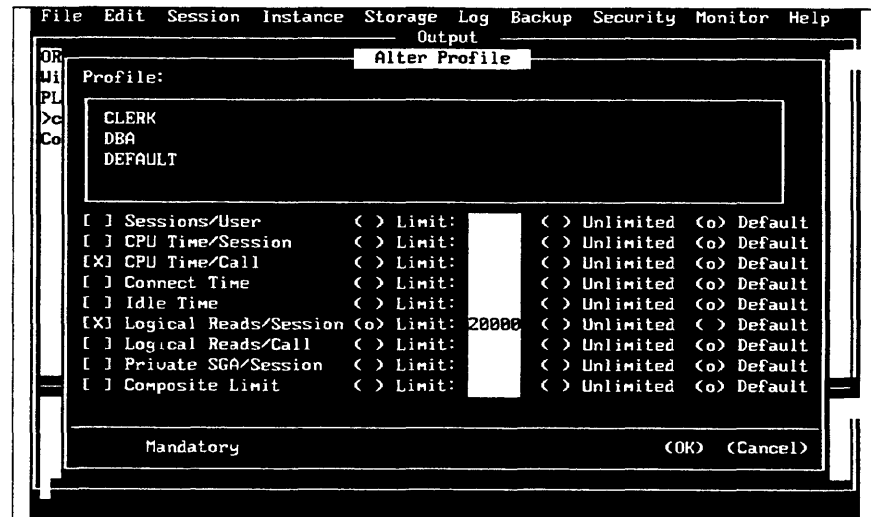
Once a profile has been created, you can assign it to database users. Each user can be assigned only one profile at any given time. If a profile is assigned to a user who already has a profile, the new profile assignment overrides the previously assigned profile. Profile assignments do not affect current sessions. Profiles can be assigned only to users and not to roles or other profiles.

Profiles can be assigned to users using the SQL*DBA dialogs Create User and Alter User, or the SQL commands CREATE USER or ALTER USER. See “Creating Users” on page 11-7 and “Altering Users” on page 11-11 for more information about assigning a profile to a user.

Altering Profiles

The resource limit settings of any profile can be altered using either the Alter Profile dialog box of SQL*DBA or the SQL command ALTER PROFILE. Figure 11-5 shows the Alter Profile dialog box.

FIGURE 11-5
The Alter Profile Dialog



The following statement is the command equivalent of the Alter Profile dialog box in Figure 11-5:

```
ALTER PROFILE clerk LIMIT
  CPU_PER_CALL default
  LOGICAL_READS_PER_SESSION 20000;
```

Any profile limit adjusted 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 (see “Using the Default Profile” on page 11-17). 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.

Privileges Required to
Alter Profiles

To alter a profile, you must have the ALTER PROFILE system privilege.

Using Composite Limits

In addition to setting specific resource limits explicitly for a profile, a single composite limit can be set that accounts for all resource limits in a profile. A profile’s composite limit is set using the Composite Limit checkbox of the Create Profile and Alter Profile dialog boxes of SQL*DBA or the COMPOSITE_LIMIT parameter of the SQL commands CREATE PROFILE or ALTER PROFILE. A composite limit is set in terms of *service units*. A service unit is a weighted sum of all resources used. For example, 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;
```

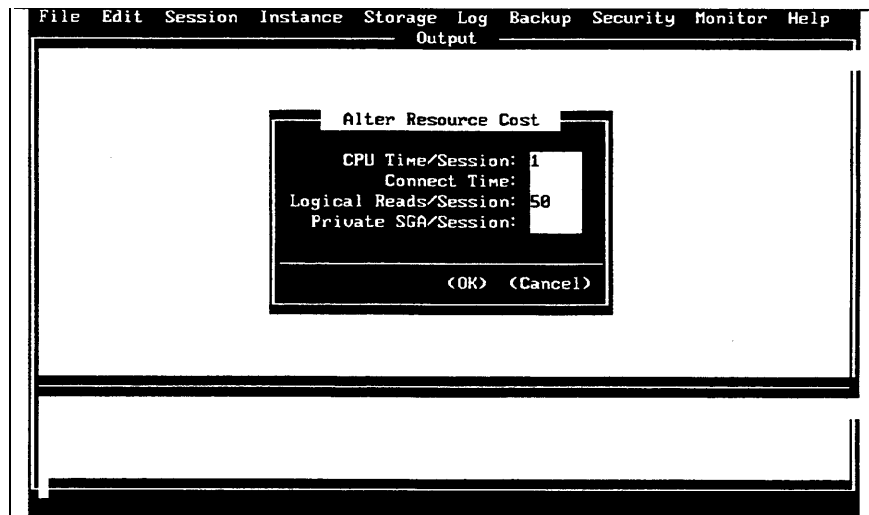
Notice that both explicit resource limits and a composite limit can exist concurrently for a profile. The limit that is reached first is used to stop the activity in a session. Composite limits allow additional flexibility when limiting the use of system resources.

The correct service unit setting for a 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.

Each system has its own characteristics; some system resources may be more valuable than others. Therefore, ORACLE also permits 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.

Only certain resources can be given a cost, including CPU_PER_SESSION, LOGICAL_READS_PER_SESSION, CONNECT_TIME, and PRIVATE_SGA. Set costs for a database using either the Alter Resource Cost dialog box of SQL*DBA or the SQL command ALTER RESOURCE COST. Figure 11-6 shows the Alter Resource Cost dialog box.

FIGURE 11-6
The Alter Resource Cost Dialog



The following statement is the command equivalent of the Alter Resource Cost dialog box in Figure 11-6:

```
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 your installation or user's guide for additional information and recommendations on setting resource costs.

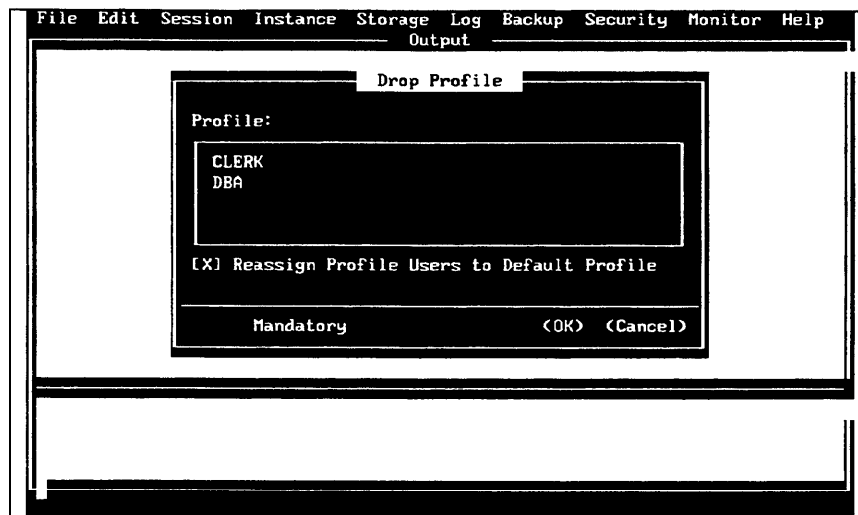
Privileges Required to Set Resource Costs

To set resource costs, you must have the ALTER RESOURCE system privilege.

Dropping Profiles

A profile can be dropped using either the Drop Profile dialog box of SQL*DBA or the SQL command DROP PROFILE. Figure 11-7 shows the Drop Profile dialog box.

FIGURE 11-7
The Drop Profile Dialog



To drop a profile currently assigned to a user successfully, the Drop Even If Currently Assigned checkbox must be selected.

The following statement is the command equivalent of the Drop Profile dialog in Figure 11-7:

```
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.

Privileges Required to
Drop Profiles

To drop a profile, you must have the DROP PROFILE system privilege.

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.

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 that an instance is started, the new parameter value is used to enable or disable 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. Once 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;
```

However, 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.

Privileges Required to Enable and Disable Resource Limits To alter the enforcement of resource limitation while the database remains open, you must have the ALTER SYSTEM system privilege.

Listing Information About Database Users and Profiles

The data dictionary stores information about every user and profile, including:

- 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 are of interest regarding database users and profiles:

- ALL_USERS
- USER_USERS
- DBA_USERS
- USER_TS_QUOTAS
- DBA_TS_QUOTAS
- USER_RESOURCE_LIMITS
- DBA_PROFILES
- RESOURCE_COST
- V\$SESSION
- V\$SESSTAT
- V\$STATNAME

See Appendix B for detailed information about each view.

Examples

The examples in this section assume a database in which the following statements have been executed:

```
CREATE PROFILE clerk LIMIT
  SESSIONS_PER_USER 1
  IDLE_TIME 30
  CONNECT_TIME 600;
```

```
CREATE USER jward
  IDENTIFIED BY wilddcat
  DEFAULT TABLESPACE users
  TEMPORARY TABLESPACE temp_ts
  QUOTA 500K ON users
  PROFILE clerk;
```

```
CREATE USER tsmith
  IDENTIFIED BY bedrock
  DEFAULT TABLESPACE users
  TEMPORARY TABLESPACE temp_ts
  QUOTA unlimited ON users;
```

Example 1 Listing All Users and Associated Information

The following query lists all users defined in the database:

```
SELECT * FROM sys.dba_users;
```

USERNA	USER_ID	PASSWORD	DEFAULT	TEMPOR	CREATED	PROFILE
-----	-----	-----	-----	-----	-----	-----
SYS	%	522D06CDE017CF93	SYSTEM	SYSTEM	31-JUL-90	PUBLIC_DEFAULT
SYSTEM	%	9B30B3EB7A7EE46A	SYSTEM	SYSTEM	31-JUL-90	PUBLIC_DEFAULT
JWARD	%	DEE4F647381D62C4	USERS	TEMP_TS	12-SEP-90	CLERK
TSMITH	%	4791F162172E7834	USERS	TEMP_TS	12-SEP-90	PUBLIC_DEFAULT

All passwords are encrypted to preserve security.

Example 2
Listing Users' Roles

The following query lists, for each user, the roles granted to that user, and indicates whether each role is granted with the ADMIN OPTION and is a default role:

```
SELECT * FROM sys.dba_role_privs where grantee = 'CCLOSKEY';
```

GRANTEE	GRANTED_ROLE	ADM	DEF
CCLOSKEY	CLERK	YES	YES
CCLOSKEY	PAYROLL	NO	NO
CCLOSKEY	WEEKLY_ADMIN	NO	NO

Example 3
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	JWARD	0	512000	0	250
SYSTEM	TSMITH	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”.

Example 4
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	LIMIT
CLERK	COMPOSITE_LIMIT	UNLIMITED
CLERK	SESSIONS_PER_USER	1
CLERK	CPU_PER_SESSION	UNLIMITED
CLERK	CPU_PER_CALL	UNLIMITED
CLERK	LOGICAL_READS_PER_SESSION	UNLIMITED
CLERK	LOGICAL_READS_PER_CALL	UNLIMITED
CLERK	IDLE_TIME	30
CLERK	CONNECT_TIME	600
CLERK	PRIVATE_SGA	UNLIMITED
DEFAULT	COMPOSITE_LIMIT	UNLIMITED
DEFAULT	SESSIONS_PER_USER	UNLIMITED
DEFAULT	CPU_PER_SESSION	UNLIMITED
DEFAULT	CPU_PER_CALL	UNLIMITED
DEFAULT	LOGICAL_READS_PER_SESSION	UNLIMITED
DEFAULT	LOGICAL_READS_PER_CALL	UNLIMITED
DEFAULT	IDLE_TIME	UNLIMITED
DEFAULT	CONNECT_TIME	UNLIMITED
DEFAULT	PRIVATE_SGA	UNLIMITED

Example 5
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'.

CONTROLLING USER PRIVILEGES AND ROLES

This chapter explains how to control the capability to execute system operations and access to schema objects using privileges and roles. Topics include:

- system and object privileges
- how to grant and revoke privileges and roles
- how to create, alter, and drop roles
- how to manage role use

For information on controlling access to a database as a whole, see Chapter 11; for suggested general database security policies, see Chapter 10.



Trusted

If you are using Trusted ORACLE in DBMS MAC mode, see the *Trusted ORACLE7 Server Administrator's Guide* for important information about system privileges and role management.

Understanding Privileges

A privilege is a right to execute a particular type of SQL statement, or a right to access another user's object. ORACLE has two kinds of privileges: system privileges and object privileges. ORACLE's privileges are listed in the following sections, along with shortcuts that group privileges commonly granted or revoked together.

System Privileges

There are over 80 distinct system privileges. Each system privilege allows a user to perform a particular database operation or class of database operations. Table 12-1 lists all system privileges and the operations that they permit.

TABLE 12-1
System Privileges

System Privilege	Operations Permitted
ANALYZE	
ANALYZE ANY	Analyze any table, cluster, or index in the database.
AUDIT	
AUDIT ANY	Audit any schema object in the database.
AUDIT SYSTEM	Enable and disable statement and privilege audit options.
CLUSTER	
CREATE CLUSTER	Create a cluster in own schema.
CREATE ANY CLUSTER	Create a cluster in any schema. Behaves similarly to CREATE ANY TABLE.
ALTER ANY CLUSTER	Alter any cluster in the database.
DROP ANY CLUSTER	Drop any cluster in the database.
DATABASE	
ALTER DATABASE	Alter the database; add files to the operating system via ORACLE, regardless of operating system privileges.
DATABASE LINK	
CREATE DATABASE LINK	Create private database links in own schema.

INDEX	
CREATE ANY INDEX	Create an index (in any schema) on any table.
ALTER ANY INDEX	Alter any index in the database.
DROP ANY INDEX	Drop any index in the database.
PRIVILEGE	
GRANT ANY PRIVILEGE	Grant any system privilege (not object privilege).
PROCEDURE	
CREATE PROCEDURE	Create stored procedures, functions, and packages in own schema.
CREATE ANY PROCEDURE	Create stored procedures, functions, and packages in any schema. (Requires that user also have ALTER ANY TABLE, BACKUP ANY TABLE, DROP ANY TABLE, LOCK ANY TABLE, COMMENT ANY TABLE, SELECT ANY TABLE, INSERT ANY TABLE, UPDATE ANY TABLE, DELETE ANY TABLE, or GRANT ANY TABLE.)
ALTER ANY PROCEDURE	Compile any stored procedure, function, or package in any schema.
DROP ANY PROCEDURE	Drop any stored procedure, function, or package in any schema.
EXECUTE ANY PROCEDURE	Execute any procedure or function (stand-alone or packaged), or reference any public package variable in any schema.
PROFILE	
CREATE PROFILE	Create profiles.
ALTER PROFILE	Alter any profile in the database.
DROP PROFILE	Drop any profile in the database.
ALTER RESOURCE COST	Set costs for resources used in all user sessions.
PUBLIC DATABASE LINK	
CREATE PUBLIC DATABASE LINK	Create public database links.
DROP PUBLIC DATABASE LINK	Drop public database links.

PUBLIC SYNONYM	
CREATE PUBLIC SYNONYM	Create public synonyms.
DROP PUBLIC SYNONYM	Drop public synonyms.
ROLE	
CREATE ROLE	Create roles.
ALTER ANY ROLE	Alter any role in the database.
DROP ANY ROLE	Drop any role in the database.
GRANT ANY ROLE	Grant any role in the database.
ROLLBACK SEGMENT	
CREATE ROLLBACK SEGMENT	Create rollback segments.
ALTER ROLLBACK SEGMENT	Alter rollback segments.
DROP ROLLBACK SEGMENT	Drop rollback segments.
SESSION	
CREATE SESSION	Connect to the database.
ALTER SESSION	Issue ALTER SESSION statements.
RESTRICTED SESSION	Connect when the database has been started using STARTUP RESTRICT. (The OSOPER and OSDBA roles contain this privilege.)
SEQUENCE	
CREATE SEQUENCE	Create a sequence in own schema.
CREATE ANY SEQUENCE	Create any sequence in any schema.
ALTER ANY SEQUENCE	Alter any sequence in any schema.
DROP ANY SEQUENCE	Drop any sequence in any schema.
SELECT ANY SEQUENCE	Reference any sequence in any schema.

SNAPSHOT	
CREATE SNAPSHOT	Create snapshots in own schema. (User must also have the CREATE TABLE privilege.)
CREATE ANY SNAPSHOT	Create snapshots in any schema. (User must also have the CREATE ANY TABLE privilege.)
ALTER ANY SNAPSHOT	Alter any snapshot in any schema,
DROP ANY SNAPSHOT	Drop any snapshot in any schema,
SYNONYM	
CREATE SYNONYM	Create a synonym in own schema.
CREATE ANY SYNONYM	Create any synonym in any schema.
DROP ANY SYNONYM	Drop any synonym in any schema
SYSTEM	
ALTER SYSTEM	Issue ALTER SYSTEM statements.
TABLE	
CREATE TABLE	Create tables in own schema. Also allows grantee to create indexes (including those for integrity constraints) on tables in own schema. (The grantee must have a quota for the tablespace or the UNLIMITED TABLESPACE privilege.)
CREATE ANY TABLE	Create a table in any schema. (If grantee has CREATE ANY TABLE privilege and creates a table in another user's schema, the owner's space quotas and default tablespace are used. If a tablespace is explicitly specified, the owner must have space quota on that tablespace. The table owner need not have the CREATE [ANY] TABLE privilege.)
ALTER ANY TABLE	Alter any table in any schema and compile any view in any schema.
BACKUP ANY TABLE	Perform an incremental export using the Export utility.
DROP ANY TABLE	Drop any table in any schema.
LOCK ANY TABLE	Lock any table or view in any schema.
COMMENT ANY TABLE	Comment on any table, view, or column in schema.
SELECT ANY TABLE	Query any table, view, or snapshot in any schema.
INSERT ANY TABLE	Insert rows into any table or view in any schema.

UPDATE ANY TABLE	Update rows in any table or view in any schema.
DELETE ANY TABLE	Delete rows from any table or view in any schema, and truncate any table, cluster, or index.
TABLESPACE	
CREATE TABLESPACE	Create tablespaces; add files to the operating system via ORACLE, regardless of the user's operating system privileges.
ALTER TABLESPACE	Alter tablespaces; add files to the operating system via ORACLE, regardless of the user's operating system privileges.
MANAGE TABLESPACE	Take any tablespace offline, bring any tablespace online, and begin and end backups of any tablespace.
DROP TABLESPACE	Drop tablespaces.
UNLIMITED TABLESPACE	Use an unlimited amount of any tablespace. (This privilege overrides any specific quotas assigned. If revoked, the grantee's schema objects remain but further tablespace allocation is denied unless allowed by specific tablespace quotas. This system privilege can only be granted to users and not to roles. In general, specific tablespace quotas are assigned instead of granting this system privilege.)
TRANSACTION	
FORCE TRANSACTION	Force the commit or rollback of own in-doubt distributed transactions in the local database.
FORCE ANY TRANSACTION	Force the commit or rollback of any in-doubt distributed transaction in the local database.
TRIGGER	
CREATE TRIGGER	Create a trigger in own schema.
CREATE ANY TRIGGER	Create any trigger (in any schema) associated with any table in any schema.
ALTER ANY TRIGGER	Enable, disable, or compile any trigger in any schema.
DROP ANY TRIGGER	Drop any trigger in any schema.

USER	
CREATE USER	Create users; assign quotas on any tablespace, set default and temporary tablespaces, and assign a profile as part of a CREATE USER statement.
BECOME USER	Become another user. (Required by any user performing a full database import.)
ALTER USER	Alter other users: change any user's password or authentication method, assign tablespace quotas, set default and temporary tablespaces, assign profiles and default roles, in an ALTER USER statement. (Not needed to alter own password)
DROP USER	Drop another user.
VIEW	
CREATE VIEW	Create a view in own schema.
CREATE ANY VIEW	Create a view in any schema. (Requires that user also have ALTER ANY TABLE, BACKUP ANY TABLE, DROP ANY TABLE, LOCK ANY TABLE, COMMENT ANY TABLE, SELECT ANY TABLE, INSERT ANY TABLE, UPDATE ANY TABLE, DELETE ANY TABLE, or GRANT ANY TABLE.)
DROP ANY VIEW	Drop any view in any schema.

Because system privileges are very powerful, they should be cautiously granted to roles and trusted users of the database.

Object Privileges

Each type of object has different privileges associated with it. Table 12-2 summarizes the object privileges available for each type of object.

TABLE 12-2
Object Privileges

Object Privilege	Table	View	Sequence	Procedure*
ALTER	✓		✓	
DELETE	✓	✓		
EXECUTE				✓
INDEX	✓**			
INSERT	✓	✓		
REFERENCES	✓**			
SELECT	✓	✓***	✓	
UPDATE	✓	✓		

* Includes stand-alone stored procedures and functions, and public package constructs.

** Privilege cannot be granted to a role.

*** Can also be granted for snapshots.

Granting INSERT Privileges for Columns

You can grant INSERT on individual columns in a table. Before granting a column-specific INSERT privilege, investigate the possible side effects; if a table has one or more columns declared NOT NULL, granting selective insert capability without including the NOT NULL columns prevents the user from inserting any rows into the table. To prevent problems, check that each column with a NOT NULL constraint is either insertable or has a non-NULL default value; if this problem is not accounted for, an error is returned and rows cannot be inserted.

Statements Permitted by
Each Privilege

Table 12-3 lists the SQL statements permitted by the object privileges listed previously.

TABLE 12-3
SQL Statements Permitted by
Database Object Privileges

Object Privilege	SQL Statements Permitted
ALTER	ALTER object (table or sequence)
DELETE	DELETE FROM object (table or view)
EXECUTE	EXECUTE object (procedure or function) References to public package variables
INDEX	CREATE INDEX ON object (tables only)
INSERT	INSERT INTO object (table or view)
REFERENCES	CREATE or ALTER TABLE statement defining a FOREIGN KEY integrity constraint on object (tables only)
SELECT	SELECT . . . FROM object (table, view, or snapshot) SQL statements using a sequence
UPDATE	UPDATE object (table or view)

Not all types of schema objects are included in Table 12-2. Many of the schema objects not listed here (such as clusters, indexes, triggers, and database links) are controlled exclusively using system privileges. For example, to alter a cluster, a user must own the cluster or have the ALTER ANY CLUSTER system privilege.

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 itself; 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.

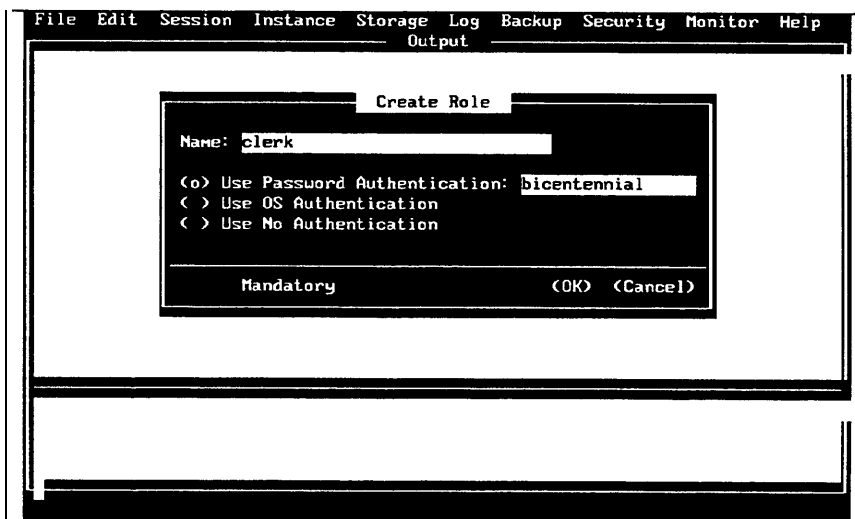
Managing Roles

A role groups several privileges and roles, so that they can be granted and revoked simultaneously from users. Roles can be enabled and disabled per user. The following sections explain how to create, alter, enable, disable, and drop roles.

Creating Roles

You create a role using either the Create Role dialog box of SQL*DBA or the SQL command CREATE ROLE. Figure 12-1 shows the Create Role dialog box.

FIGURE 12-1
The Create Role Dialog



The following statement is the command equivalent of the Create Role dialog box in Figure 12-1:

```
CREATE ROLE clerk IDENTIFIED BY bicentennial;
```

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.

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. This procedure is discussed in “Granting and Revoking Privileges and Roles” on page 12-15.

Role Names in Multi-Byte Character Sets

In a database that uses a multi-byte character set, Oracle Corporation 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.

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) or, if your operating system permits, by the operating system.

Authorization by the Database The use of a role can be protected by an associated password, as in the examples above. 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 command.

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.

Authorization by the Operating System In contrast, 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 must be 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 to be 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 (that is, if OS_ROLES = TRUE; see page 12-22), you do not need to have the operating system authorize them also; this is redundant.

Note: If users connect to the database through the multi-threaded server, by default their roles cannot be authenticated by the operating system. 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 with the multi-threaded server, 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.)

No 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.

Using Pre-Defined Roles

The roles listed in Table 12-4 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 pre-defined roles, as you can to any role you define.

TABLE 12-4
Pre-Defined 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
RESOURCE ^{1,2}	CREATE CLUSTER, CREATE PROCEDURE, CREATE SEQUENCE, CREATE TABLE, CREATE TRIGGER
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, WRITEDOWN ⁶

- 1 created by SQL.BSQ
- 2 grantees of the RESOURCE role also receive the UNLIMITED TABLESPACE system privilege as an explicitly grant (not as part of the RESOURCE role)
- 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)
- 4 also includes the EXP_FULL_DATABASE and IMP_FULL_DATABASE roles if CATEXP.SQL has been run
- 5 created by CATEXP.SQL
- 6 a Trusted ORACLE privilege only; see the *Trusted ORACLE7 Server Administrator's Guide*



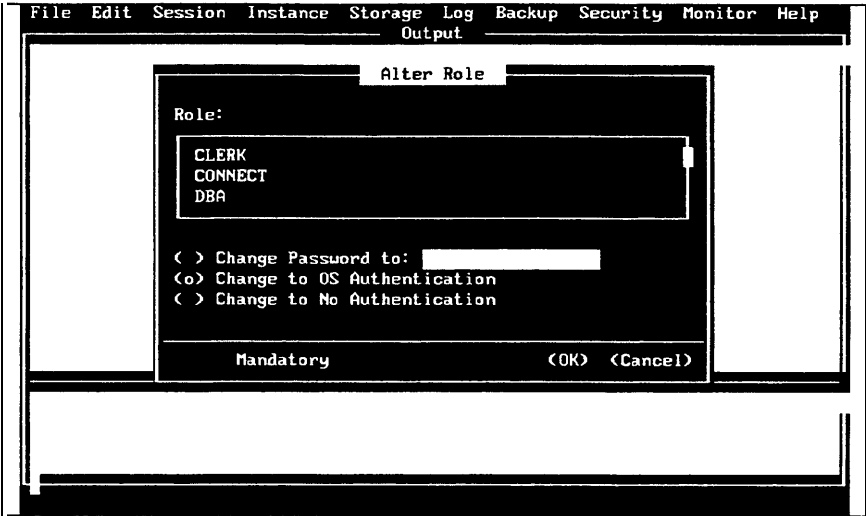
Privileges Required to
Create Roles

To create a role, you must have the CREATE ROLE system privilege. Typically, only security administrators have this system privilege.

Changing Role
Authorization

You can set and change the authorization method for a role using either the Alter Role dialog box of SQL*DBA or the SQL command ALTER ROLE. Figure 12-2 shows the Alter Role dialog box.

FIGURE 12-2
The Alter Role Dialog



The following statement is the command equivalent of the Alter Role dialog box in Figure 12-2.

```
ALTER ROLE clerk IDENTIFIED EXTERNALLY;
```

For more information about role authorization, see “Role Authorization” on page 12-11.

Privileges Required to
Change Role
Authorization

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.

**Changing a User’s
Default Roles**

A user’s list of default roles can be set and altered using either the Alter User dialog box of SQL*DBA or the SQL command ALTER USER. See “Altering Users” on page 11-11 for more information about these options.

If the user’s list of default roles is specified as ALL, every role granted to a user is automatically added to the user’s list of default roles. Only subsequent modification of a user’s default role list can remove newly granted roles from a user’s list of default roles.

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 four 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.

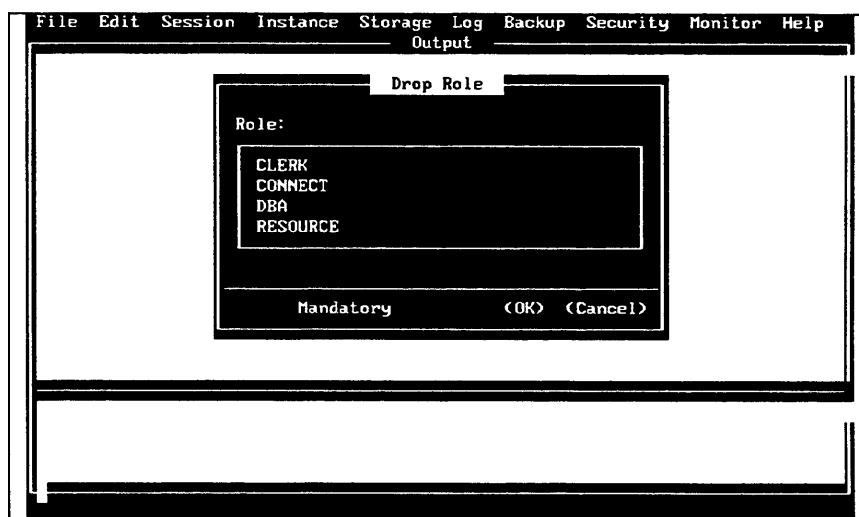
Dropping Roles

In some cases, it may be applicable 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.

Drop a role using either the Drop Role dialog box of SQL*DBA or the SQL command DROP ROLE. Figure 12-3 shows the Drop Role dialog box.

FIGURE 12-3
The Drop Role Dialog



The following statement is the command equivalent of the Drop Role dialog box in Figure 12-3:

```
DROP ROLE clerk;
```

Privileges Required to Drop Roles

To drop a role, you must have the DROP ANY ROLE system privilege or have been granted the role with the ADMIN OPTION.

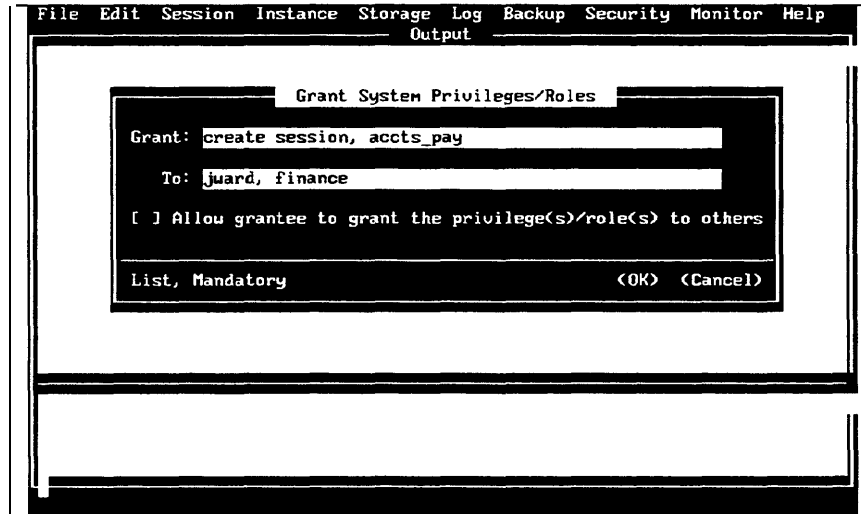
Granting and Revoking Privileges and Roles

This section explains how to grant and revoke system privileges, roles, and object privileges.

Granting System Privileges and Roles

System privileges and roles can be granted to other roles or users using either the Grant System Privileges/Roles dialog box of SQL*DBA or the SQL command GRANT. Figure 12-4 shows the Grant System Privileges/Roles dialog box.

FIGURE 12-4
The Grant System Privileges/
Roles Dialog



The following statement is the command equivalent of the Grant System Privileges/Roles dialog box in Figure 12-4:

```
GRANT create session, accts_pay  
TO jward, finance;
```

Note: Object privileges **cannot** be granted along with system privileges and roles in the same GRANT statement.

The ADMIN Option

A system privilege or role can be granted with the ADMIN OPTION. (The ADMIN OPTION cannot be included in a GRANT statement that grants a role to another role.) A grantee with this special 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. (A user cannot revoke a role from himself.)

- 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.

A grantee without the ADMIN OPTION cannot perform the above operations.

When a user creates a role, the role is automatically granted to the creator with the ADMIN OPTION.

Assume that the security administrator grants the NEW_DBA role to MICHAEL with the following statement:

```
GRANT new_dba TO michael WITH ADMIN OPTION;
```

The user MICHAEL cannot only use all of the privileges implicit in the NEW_DBA role, but can grant, revoke, or drop the NEW_DBA role as he deems necessary. Because of these powerful capabilities, use caution when granting system privileges or roles with the ADMIN OPTION. Such capabilities are usually reserved for a security administrator and rarely given to other administrators or users of the system.

Privileges Required to
Grant System Privileges or
Roles

To grant a system privilege or role, the grantor requires the ADMIN OPTION for all system privileges and roles being granted. Additionally, any user with the GRANT ANY ROLE system privilege can grant any role in a database.

Granting Object Privileges

Object privileges can be granted to roles or users using the SQL command GRANT. For example, the following statement grants the SELECT, INSERT, and DELETE object privileges for all columns of the EMP table to the users JWARD and TSMITH:

```
GRANT select, insert, delete ON emp TO jward, tsmith;
```

To grant the INSERT object privilege for only the ENAME and JOB columns of the EMP table to the users JWARD and TSMITH, enter the following statement:

```
GRANT insert (ename, job) ON emp TO jward, tsmith;
```

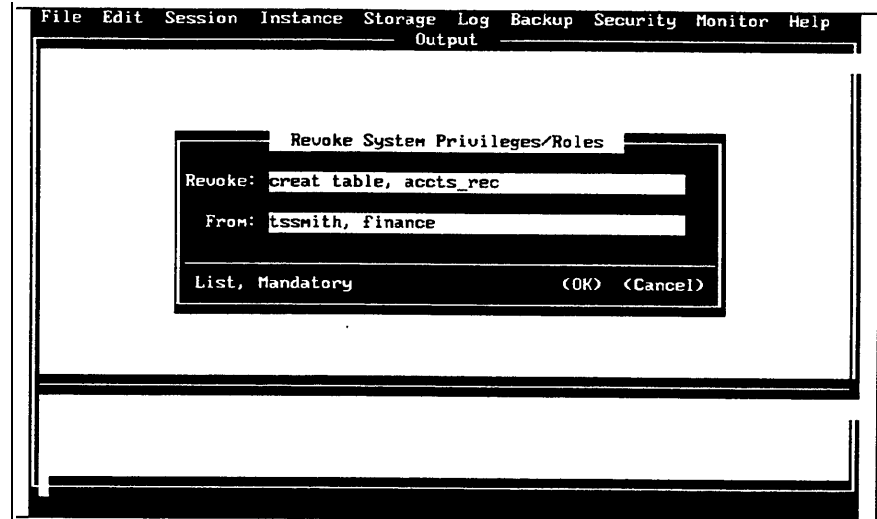
To grant all object privileges on the SALARY view to the user WALLEN, use the ALL shortcut, as in:

```
GRANT ALL ON salary TO wallen;
```

Note: System privileges and roles cannot be granted along with object privileges in the same GRANT statement.

The GRANT OPTION	An object privilege can be granted to a user with the GRANT OPTION. This special privilege allows the grantee several expanded privileges: • The grantee can grant the object privilege to any user or any role in the database. • The grantee can also grant the object privilege to other users, with or without the GRANT OPTION. • If the grantee receives object privileges for a table with the GRANT OPTION and the grantee has the CREATE VIEW or the 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 user whose schema contains an object is automatically granted all associated object privileges with the GRANT OPTION. Specifically note that 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 via roles.
Privileges Required to Grant Object Privileges	To grant an object privilege, the grantor fulfill one of the following conditions: • Own the object specified. • Have been granted the object privileges being granted with the GRANT OPTION.
Revoking System Privileges and Roles	System privileges and/or roles can be revoked using either the Revoke System Privileges/Roles dialog box of SQL*DBA or the SQL command REVOKE. Figure 12-5 shows the Revoke System Privileges/Roles dialog box.

FIGURE 12-5
**The Revoke System Privileges/
 Roles Dialog**



The following statement is the command equivalent of the Revoke System Privileges/Roles dialog box in Figure 12-5:

```
REVOKE create table, accts_rec FROM tsmith, finance;
```

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 regranted without the ADMIN OPTION.

Privileges Required to Revoke System Privileges and Roles

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 user does not have to be the user that originally granted the privilege or role). Additionally, any user with the GRANT ANY ROLE can revoke any role.

Revoking Object Privileges

Object privileges can be revoked using the SQL command REVOKE. For example, assuming you are the original grantor, to revoke the SELECT and INSERT privileges on the EMP table from the users JWARD and TSMITH, enter the following statement:

```
REVOKE select, insert ON emp  

  FROM jward, tsmith;
```

A grantor could also revoke all privileges on the table DEPT (even if only one privilege was granted) that he/she granted to the role HUMAN_RESOURCES by entering the following statement:

```
REVOKE ALL ON dept FROM human_resources;
```

Note: The 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 regranted 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 regrant 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 enter 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 regrants 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 only revoke the privilege 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.

Privileges Required to Revoke Object Privileges

To revoke an object privilege, the revoker must be the original grantor of the object privilege being revoked.

Cascading Effects of Revoking Privileges

Depending on the type of privilege, there may or may not be cascading effects if a privilege is revoked. The following sections explain several cascading effects.

System Privileges

There are no cascading effects when revoking a system privilege related to DDL operations, regardless of whether or not the privilege was granted with or without the ADMIN OPTION. For example, assume:

1. The security administrator grants the CREATE TABLE system privilege to JWARD with the ADMIN OPTION.
2. JWARD creates a table.
3. JWARD grants the CREATE TABLE system privilege to TSMITH.
4. TSMITH creates a table.
5. The security administrator revokes the CREATE TABLE system privilege from JWARD.
6. JWARD's table continues to exist. TSMITH continues to have the CREATE TABLE system privilege and his table still exists.

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 reauthorized before they can be used again (after the revoke).

Object Privileges

Revoking an object privilege can have several types of cascading effects that should be investigated before a REVOKE statement is issued:

- 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 through the use of 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 only grant a privilege or role to PUBLIC if every database user requires the privilege or role. This recommendation restates the general rule that at any given time, each database user should only have the privileges required to accomplish the current task successfully.

Revokes from PUBLIC can cause significant cascading effects, depending on the privilege that is revoked. 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* (see "Managing Object Dependencies" on page 8-75) before they can be used again. Therefore, use caution when granting DML-related privileges to PUBLIC.

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 privileges (system and object) 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 Database Roles Using the Operating System

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, as normal.

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 if the operating system is managing roles. However, you can change this default; see "Using the Multi-Threaded Server with Operating System Role Management" on page 12-24.



Note: The features described in this section are available only on some operating systems. This information is operating system dependent; see your installation or user's guide.

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	ID specifies different things 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.

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 the Multi-Threaded Server 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, 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.)

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

Note: See Appendix B for a detailed description of these data dictionary views.

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;
```


Example 1
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

Example 2
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

Example 3
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, the following query can be used:

```
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	ENAME	INSERT
JWARD	EMP	JOB	INSERT

Example 4
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, the following results are returned:

```
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.

Example 5
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

Example 6
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

AUDITING DATABASE USE

This chapter explains how to use the ORACLE auditing facilities.
Topics include:

- guidelines for auditing your database
- setting auditing options
- turning auditing on and off
- managing the audit trail
- auditing with database triggers



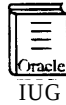
Trusted

If you are using Trusted ORACLE, see the *Trusted ORACLE7 Server Administrator's Guide* for additional information about auditing and audit trail management.

Guidelines for Auditing

Here are some guidelines for auditing database activity.

Choose Between Auditing with the Database and Auditing with the Operating System



The data dictionary of every database has a table named SYS.AUD\$, commonly referred to as the database *audit trail*. 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 installation or user's guide). 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 SQL*ReportWriter) 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.

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 also minimize the size of the audit trail.

Consider the following two general guidelines when devising an auditing strategy:

1. Evaluate your purpose for auditing.

Once 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.

2. **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 your auditing purpose is to monitor for suspicious database activity, consider 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 "Protecting the Audit Trail" on page 13-17.

Auditing Normal Database Activity

When your auditing purpose is to gather historical information about particular database activities, consider 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.**

Once 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

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.



On most operating systems (see your installation or user's guide for more information), the audit trail views are created automatically with the data dictionary. They must be created for you to use auditing; you can later delete them if you decide not to use auditing. The following sections explain how to do this.

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 Appendix B for more information about these views. Also see “Viewing Database Audit Trail Information” on page 13-17 for examples of interpreting audit information.

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.

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; see page 13-20 for examples of using triggers for this specialized type of auditing.

ORACLE allows you to set audit options at three levels:

statement	audits based on the type of a SQL statement, 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

The following sections explain the audit options you can set.

Statement Audit Options

Valid statement audit options that can be included in AUDIT and NOAUDIT statements are listed in Table 13-1. (Additional statement audit options are listed in Table 13-2.)

TABLE 13-1
Statement Auditing Options

Option	SQL Statements Audited
ALTER SYSTEM	ALTER SYSTEM
CLUSTER	CREATE CLUSTER ALTER CLUSTER TRUNCATE CLUSTER DROP CLUSTER
DATABASE LINK	CREATE DATABASE LINK DROP DATABASE LINK
INDEX	CREATE INDEX ALTER INDEX DROP INDEX
NOT EXISTS	All SQL statements that return an ORACLE error because the specified structure or object does not exist
PROCEDURE	CREATE [OR REPLACE] FUNCTION CREATE [OR REPLACE] PACKAGE CREATE [OR REPLACE] PACKAGE BODY CREATE [OR REPLACE] PROCEDURE DROP FUNCTION DROP PACKAGE DROP PROCEDURE
PUBLIC DATABASE LINK	CREATE PUBLIC DATABASE LINK DROP PUBLIC DATABASE LINK
PUBLIC SYNONYM	CREATE PUBLIC SYNONYM DROP PUBLIC SYNONYM
ROLE	CREATE ROLE ALTER ROLE SET ROLE DROP ROLE
ROLLBACK SEGMENT	CREATE ROLLBACK SEGMENT ALTER ROLLBACK SEGMENT DROP ROLLBACK SEGMENT
SEQUENCE	CREATE SEQUENCE DROP SEQUENCE
SESSION	Connects and Disconnects
SYNONYM	CREATE SYNONYM DROP SYNONYM
SYSTEM AUDIT	AUDIT NO AUDIT

SYSTEM GRANT	GRANT system privilege/role TO user/role REVOKE system privilege/role FROM user/role
TABLE	CREATE TABLE TRUNCATE TABLE DROP TABLE
TABLESPACE	CREATE TABLESPACE ALTER TABLESPACE DROP TABLESPACE
TRIGGER	CREATE TRIGGER ALTER TRIGGER ENABLE or DISABLE ALTER TABLE with ENABLE, DISABLE, and DROP clauses
USER	CREATE USER ALTER USER DROP USER
VIEW	CREATE [OR REPLACE] VIEW DROP VIEW

Shortcuts for Statement Audit Options



Shortcuts are provided to 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.

CONNECT	equivalent to the SESSION option
RESOURCE	equivalent to the options ALTER SYSTEM, CLUSTER, DATABASE LINK, PROCEDURE, ROLLBACK SEGMENT, SEQUENCE, SYNONYM, TABLE, TABLESPACE, and VIEW
DBA	equivalent to the options SYSTEM AUDIT, PUBLIC DATABASE LINK, PUBLIC SYNONYM, ROLE, SYSTEM GRANT, and USER
ALL	equivalent to all options in Table 13-1, including the NOT EXISTS option



Note: Do not confuse the shortcuts CONNECT, RESOURCE, and DBA with the predefined roles of the same names.

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.

Additional Statement
options

Additional audit options not covered by any of the above shortcuts are listed in Table 13-2.

TABLE 13-2
Additional Statement
Auditing Options

Object Option	SQL Statements Audited
ALTER SEQUENCE	ALTER SEQUENCE sequence
ALTER TABLE	ALTER TABLE table
COMMENT TABLE	COMMENT ON table, view, snapshot, column
DELETE TABLE	DELETE FROM table, view
EXECUTE PROCEDURE	Calls to procedures and functions
GRANT PROCEDURE	GRANT privilege ON procedure REVOKE privilege ON procedure
GRANT SEQUENCE	GRANT privilege ON sequence REVOKE privilege ON sequence
GRANT TABLE	GRANT privilege ON table, view, snapshot REVOKE privilege ON table, view, snapshot
INSERT TABLE	INSERT INTO table, view
LOCK TABLE	LOCK TABLE table, view
SELECT SEQUENCE	Reference to a sequence
SELECT TABLE	SELECT . . . FROM table, view, snapshot
UPDATE TABLE	UPDATE table, view

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 like the following example:

```
AUDIT DELETE ANY TABLE
  BY ACCESS
  WHENEVER NOT SUCCESSFUL;
```

ORACLE's system privileges are listed beginning on page 12-2.

Object Audit Options

Table 13-3 lists valid object audit options and the schema object types for which each option is available.

TABLE 13-3
Object Audit Options

Object Option	Table	View	Sequence	Procedure*
ALTER	✓		✓	
AUDIT	✓	✓	✓	✓
COMMENT	✓	✓		
DELETE	✓	✓		
EXECUTE				✓
GRANT	✓	✓	✓	✓
INDEX	✓			
INSERT	✓	✓		
LOCK	✓	✓		
RENAME	✓	✓		✓
SELECT	✓	✓**	✓	
UPDATE	✓	✓		

* “Procedure” refers to stand-alone stored procedures and functions, and packages.

** The SELECT option may also be used for snapshots.

Table 13-4 lists the SQL statements audited by each object option.

TABLE 13-4
SQL Statement Audited by
Database Object Audit Options

Object Option	SQL Statements Audited
ALTER	ALTER object (table or sequence)
AUDIT	AUDIT (Form II) object
COMMENT	COMMENT object (table or view)
DELETE	DELETE FROM object (table or view)
EXECUTE	EXECUTE object (procedure*)
GRANT	GRANT (Form II) privilege ON object
INDEX	CREATE INDEX ON object (tables only)
INSERT	INSERT INTO object (table or view)
LOCK	LOCK object (table or view)
RENAME	RENAME object (table, view, or procedure*)
SELECT	SELECT . . . FROM object (table, view, snapshot) Any statement using a sequence
UPDATE	UPDATE object (table or view)

* Procedure refers to stand-alone stored procedures and functions, and packages.

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 command AUDIT turns on statement and privilege audit options, and object audit options. Audit statements that set 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. The SQL command AUDIT turns on 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.

You can set any auditing option, and specify the following conditions for auditing:

- WHENEVER SUCCESSFUL / WHENEVER NOT SUCCESSFUL
- BY SESSION / BY ACCESS



Warning: The AUDIT command only turns auditing options on; 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 "Enabling and Disabling Database Auditing" on page 13-14.)



The following examples illustrate the use of the AUDIT command. For a complete description of the AUDIT command, see the *ORACLE7 Server SQL Language Reference Manual*.

Enabling Statement and 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 ANY PROCEDURE system privilege, by all database users, BY ACCESS, enter the following statement:

```
AUDIT SELECT TABLE, INSERT TABLE, DELETE TABLE,
  EXECUTE ANY PROCEDURE
  BY ACCESS
  WHENEVER NOT SUCCESSFUL;
```

Privilege Required to Enable Statement and Privilege Auditing

Options 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 EMP table, BY SESSION (the default value), enter the following statement:

```
AUDIT DELETE ON 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;
```

Setting Default Object Audit Options 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;
```

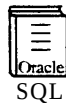
Privilege Required to Enable Object Audit Options A user can set any object audit option for the objects contained in the user's schema. The AUDIT ANY system privilege is required to set an object audit option for an object contained in another user's schema or to set the default object auditing options; 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 non-successful 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.



The following examples illustrate the use of the NOAUDIT command. For a complete syntax listing of the NOAUDIT command, see the *ORACLE7 Server SQL Language Reference Manual*.



Warning: The NOAUDIT command only turns auditing options off; 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 "Enabling and Disabling Database Auditing" on page 13-14.)

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 ANY PROCEDURE;
```

Furthermore, the following statements turn off all statement (system) and privilege audit options:

```
NOAUDIT ALL;
NOAUDIT ALL PRIVILEGES;
```

Privileges Required to Disable Statement and Privilege Auditing

Options 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.

Privileges Required to Disable Object Auditing Options 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.)

Once you have edited the parameter file, restart the database instance to enable or disable database auditing as intended.

Note: See Appendix A for more information about editing parameter files.

Controlling the Growth and Size of the Audit Trail



Note: The following sections discuss how to manage the database audit trail. If you are directing audit records to the operating system audit trail, see your installation or user's guide for more information about managing the operating system 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.

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



Utilities

system file. For information about exporting tables, see the *ORACLE7 Server Utilities User's Guide*.

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.

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 INTERNAL.
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 that 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 itself, audit the audit trail with 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 as INTERNAL, which itself has protection against unauthorized use. As a final measure of protecting the audit trail, any operation performed while connected as INTERNAL is audited in the operating system audit trail, if available. See your installation or user's guide for more information about the availability of an operating system audit trail and possible uses.



Viewing Database Audit Trail Information

The following sections give examples that demonstrate how to examine and interpret the information in the audit trail.

Examples Assume that you must audit the database for the following suspicious database activity:

- 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.

Also assume that you suspect the users JWARD and SWILLIAMS for several of these detrimental actions.

For the examples below, assume that the following statements are issued by an administrator (in the given order):

```

AUDIT ALTER, INDEX, RENAME ON DEFAULT
    BY SESSION;
CREATE TABLE scott.emp . . . ;
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.

Example 1
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 or not they are set for success or failure (or both), and whether they are set for BY SESSION or BY ACCESS.

Example 2
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	AUDIT_OPTION	SUCCESS	FAILURE
-----	-----	-----	-----
	ALTER USER	BY SESSION	BY SESSION

Example 3
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	REN	SEL	UPD	EXE
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----
SCOTT	EMP	TABLE	S/S	-/-	-/-	A/-	-/-	S/S	-/-	-/-	S/S	-/-	-/-	-/-
SCOTT	EMPLOYEE	VIEW	-/-	-/-	-/-	-/-	-/-	-/-	-/-	-/-	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.

Example 4
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
---	---	---	---	---	---	---	---	---	---	---
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).

Example 5
Listing Audit Records

The following query lists audit records generated by statement and object audit options:

```
SELECT username, obj_name, action_name, ses_actions
FROM sys.dba_audit_object;
```

Example 6
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 or not 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. In contrast, triggers can audit only DML statements issued against tables.
- 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 vs. 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:

- 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.

PART

V

**DISTRIBUTED
PROCESSING AND
DISTRIBUTED
DATABASES**

14

ADMINISTERING CLIENT-SERVER SYSTEMS

This chapter provides guidelines for managing client-server systems. The following chapters are included:

- placing distributed applications
- developing distributed applications
- connecting between versions of ORACLE

Deciding Where to Locate Client Applications

In a distributed processing environment, you must work with application developers to decide the best location for the client applications. Below are examples of making the proper choice for this location.

- ORACLE is operating on a mainframe computer. The majority of the database users are unfamiliar with mainframe computers; however, they are familiar with personal computers (PCs). To avoid having to educate users on the mainframe's operating system, you might find it easiest to operate the client application code on the local PC and not on the mainframe.
- ORACLE is operating on a mainframe computer. The majority of the database users have PCs locally on their desk top. To boost ORACLE's performance and minimize network traffic, you should operate client applications on the local PCs.
- ORACLE is operating on a minicomputer. All database users have local terminals connected to the centralized minicomputer. The system will see best performance if you execute ORACLE on one processor of the minicomputer, while all client applications are executed by a different processor of the minicomputer. This configuration uses true distributed processing.

Developing Distributed Database Applications

In a distributed processing environment, you must work with the application developer to decide the best location for database applications to be developed and tested. For example, assume the following conditions:

- A company operates ORACLE on a centralized minicomputer.
- Client applications are also executed by the minicomputer. Database users interact with the database via terminals.
- At all times of the day, there is a high throughput of database transactions.

In this configuration, using the minicomputer to develop new database applications might significantly harm the performance of the database system. Instead, developers should develop and test new applications on another system running ORACLE, such as a minicomputer or a PC. Because ORACLE and applications developed for ORACLE are

portable, the finished application can then be ported to the minicomputer without changes to the existing environment.

Connecting Between ORACLE Server Versions

In administering a distributed processing network, you must be aware of the version of Oracle software running at networked sites:

- Version 5 client applications can query Version 6 servers and ORACLE7 Servers.
- Version 6 clients can query Version 5 and Version 6 servers, and ORACLE7 Servers; however, Version 6 clients cannot use features unique to Version 6 when communicating with a Version 5 server since the server does not support them.
- ORACLE7 clients can query and update Version 5 and 6 databases, with only one updated database per transaction; however, ORACLE7 clients cannot use features unique to ORACLE7 when communicating with a Version 5 or Version 6 server, since the older versions do not support them.

Attempting to update a Version 5 or 6 database and either an ORACLE7 database or a different Version 5 or 6 database results in an error message (“ORA-02047: cannot join distributed transaction in progress”).

MANAGING DISTRIBUTED DATABASES

This chapter explains how to manage a database that is part of a distributed database system. Topics include:

- guidelines for distributed databases
- managing global database names
- creating, using, and dropping database links
- implementing location transparency
- setting commit point strength
- troubleshooting distributed database problems

The information in this chapter applies only to systems using ORACLE with the distributed option.

Guidelines for Implementing a Distributed Database System

The following sections discuss some important issues that you should consider in implementing a distributed database system.

Assign Distributed Database Administrators

Because nodes in an ORACLE distributed database system are autonomous, one or more database administrators must assume responsibility for each database in a distributed database system. Each database administrator is responsible for issues important only to the associated database, such as space management and local security. However, administrators of different databases in a distributed system also need to coordinate with each other on issues that concern distributed database systems, such as data design issues, backup and recovery issues, and remote security.

Consider Network and Communication Issues



Database, system, and network administrators can be responsible for guaranteeing that networks connect machines referenced in distributed transactions. This includes installing and executing the proper SQL*Net drivers to facilitate communication among ORACLE database servers. For more information about SQL*Net, see your SQL*Net documentation.

Define Global Database Names and Database Links

Give each database a unique global name and define database links in the local database so that SQL statements issued by users and applications can reference objects of remote databases. See “Managing Global Database Names” and “Managing Database Links” on pages 15-3 and 15-7 for details.

Decide Where to Locate the Data

In a distributed database environment, database administrators must work with application developers to decide the best location for the data. Consider the following items:

- the number of transactions posted from each location
- the amount of data (portion of table) used by each node
- the performance characteristics and reliability of the network
- the speed of various nodes and the capacities of disks
- the criticality of access if a node or link is down
- the need for referential integrity among tables



For more information on trade-offs in locating data, see the *ORACLE7 Application Developer's Guide*.

Enforce Security in a Distributed Database System

In general, security is controlled by the administrator of each database and cannot be administered remotely; for example, you cannot grant system and object privileges to remote users. A remote connection is subject to the security domain of the account a database link uses for the connection. In some cases, you can perform limited local privilege management for remote objects; see “Balancing Location Transparency and Security” on page 15-12.

Resource limits for a user apply only at the node to which a user is directly connected. Remote sessions established on behalf of database links are also subject to resource limits set in the remote database.

Similarly, audit records are generated on behalf of connections established in each database, including connections established via database links. Audit records are generated in the database in which a change occurs.

Plan Distributed Database Backup and Recovery

Consider backup and recovery strategies for the global distributed database before making autonomous decisions for your databases. Several important issues are discussed in the sections “Consider Distributed Database Backups” and “Coordinate Distributed Recovery” on pages 18-4 and 19-3, respectively.

Managing Global Database Names

In a distributed database system, each ORACLE database must have a unique global database name so that objects within the distributed database can be uniquely identified. A database's global database name is comprised of two parts: a name component (used for local administrative operations such as startup, recovery operations, and shutdown) and a network domain component (used to indicate the database's location within a network structure).

Note: Although global database names resemble SQL*Net communities, which similarly have domains, they are not the same thing. See your SQL*Net documentation for information on SQL*Net communities.

Choosing a Global Database Name

Before creating an ORACLE database (or renaming an existing database), choose a global name for the database so that it is uniquely

identified among any network of databases; that is, make sure that the combination of database name and domain is unique. Consider the following points when naming a database:

- The name component of a global database name usually corresponds to the information within the database. For example, if a database stores sales information, the database might be named SALES; if a database stores archeological information about prehistoric reptiles, the database might be named DINOSAUR.
- The name component of a global database name should not imply location. This practice is usually avoided to maintain an extra layer of location transparency with respect to global object names; this transparency is lost if users explicitly specify global object names in their SQL statements.
- You should have some administrative control over the network domain that contains the database. For example, you should not create a database in a high-level network domain in which you have no control over name duplication. By having control of the network domain that contains a database, you can ensure that all database names within the domain are unique.
- Anticipate potential database name conflicts within network domains. If you foresee name conflicts within a particular domain, use an additional level in the network domain structure so that no name conflicts occur. However, create only enough levels of domains to make names unique, without creating unnecessarily long and confusing database names.
- Put databases that are related in the same network domain. This improves organization within network domains, and reduces complexity when specifying names for the database links among related databases.

Creating a Global Database Name

Specify both components of a database's global name in the parameter file before database creation; after database creation, changing the database's global name can have a severe impact on existing applications and object definitions.

You establish a database's global name by setting the DB_NAME and DB_DOMAIN parameters. The DB_NAME parameter determines the name component of a database's global object name; the DB_NAME parameter also determines the database's local database name, which you might specify during administrative operations (such as database startup and recovery). The DB_DOMAIN parameter indicates the location (strictly a logical location) of the database within a network

structure. The combination of the settings for these two parameters forms a database name that must be unique within a network. For example, to create a database with a global database name TEST.US.ACME.COM, edit the parameters of the parameter file as follows:

```
DB_NAME = TEST
DB_DOMAIN = US.ACME.COM
```

DB_NAME must be an alphanumeric text string of no more than eight characters, without any special characters. During database creation, the name you provide for DB_NAME is recorded in the data files, redo log files, and control file of the database. If during database startup the value of the DB_NAME parameter and the database name in the control file do not match, the database does not start.

DB_DOMAIN must be an alphanumeric text string of no more than 119 characters that specifies the network domain in which the database is created (typically the name of the organization that owns the database). Case is not significant. Levels in the domain name are separated by dots; dots should only separate leafs in the domain structure. The order of domain names is from leaf to root, left to right (in accordance with the standard Internet convention).

Changing a Database's Global Name

In rare situations, you may need to change a database's global name. Changing a database's global name can cause several undesired side effects, including the following effects:

- If existing database links in remote databases point to the renamed database, they are not renamed automatically.
- Existing synonyms in remote databases that point to objects within the renamed database are not updated automatically.
- Existing PL/SQL program units in remote databases that reference data in the renamed database are not updated automatically.

If you absolutely need to change the global name of a database, first coordinate the name change with other database administrators of the system. After this is completed, you should (but are not required to) alter the instance to restricted session mode (using the ENABLE RESTRICTED SESSION option of the SQL command ALTER SYSTEM) and then kill all current sessions. To rename the database, use the RENAME GLOBAL_NAME parameter of the ALTER DATABASE command. For example, assume that during a reorganization of the ACME company, the Sales department is moved to Australia. You

could rename the global database name of the Sales department database as follows:

```
ALTER DATABASE RENAME GLOBAL_NAME TO sales.australia.acme.com;
```

After the database name has been changed, you can allow users to connect to the instance; use the `DISABLE RESTRICTED SESSION` option of the SQL command `ALTER SYSTEM`.

Enforcing Global Naming

In most cases, you should take advantage of the global object naming features of ORACLE when operating in a distributed database. However, for backward compatibility with earlier releases of ORACLE, you can disable global name resolution. Disabling global name resolution also allows database link names to have no correspondence to global database names.

Enforcement of global name resolution is controlled at two levels:

for the instance	Enforcement of global name resolution can be controlled at the system level by the initialization parameter <code>GLOBAL_NAMES</code> ; if set to <code>TRUE</code> , a subsequently started instance enforces global name resolution. The default is <code>FALSE</code> . The current instance setting can be overridden using the <code>SET GLOBAL_NAMES</code> parameter of the <code>ALTER SYSTEM</code> command (specifying <code>TRUE</code> or <code>FALSE</code>).
for a user session	Enforcement of global name resolution can be controlled at the session level using the <code>SET GLOBAL_NAMES</code> parameter of the <code>ALTER SESSION</code> command (specifying <code>TRUE</code> or <code>FALSE</code>).

Examining Global Database Names

Any user can see the global name of a database with the `GLOBAL_NAME` data dictionary view. For example, the following query lists the global database name of the database to which you are connected (in this case, `SALES.US.ACME.COM`):

```
SELECT * FROM GLOBAL_ NAME;
```

```
GLOBAL_NAME
-----
SALES.US.ACME.COM
```

Managing Database Links

The database and application administrators of a database that is part of a distributed database are responsible for managing the necessary database links that define paths to related remote databases. The following sections cover the guidelines and procedures to consider when managing database links.

Guidelines for Database Links

In a distributed database, application developers and individual users are often allowed to create private database links. However, you must account for the extra security responsibilities required in a distributed system.

As the administrator, you are responsible for creating all public and network database links. In general, public links are defined so that the following conditions are true:

- All users of the local database can access data in remote databases of the system, providing they have the necessary authorizations in the remote databases.
- Users who create matching private database links do not have to be concerned with specifying complicated database strings.

Application developers and end-users do not need to know anything about SQL*Net connect strings and fully specified global database names, but instead need just a list of remote database names. Using this approach allows you to control how private database links establish connections to remote databases.



Concepts

Tip: Take advantage of the fact that ORACLE can create a full connect string from partial links to make database link management simple and well-defined. Create public links with database strings for all related databases, so that application developers and end-users can create simple private links without having to specify network connection information. See the *ORACLE7 Server Concepts Manual* for details on ORACLE database link search paths.

Creating Database Links

Users and administrators can create database links to facilitate connections to remote databases when a global object name is referenced in a SQL statement or PL/SQL program unit. The name of a database link must be the same as that of the database to which it points. If there is no such database, the link is created, but returns errors when used to resolve object references.



To create a database link and define a path to a remote database, use the `CREATE DATABASE LINK` command. The optional `CONNECT TO` clause specifies the remote username/password to use when creating a session in the remote database; if this clause is omitted, the local username and password are used to create the remote ORACLE user session. The optional `USING` clause indicates the database string (a string containing operating system specific and communications-specific SQL*Net connection information that identifies the communications protocol). See your installation or user's guide and SQL*Net documentation for more information on connect strings.

For example, assume that the local database is named `HQ.ACME.COM` and a remote database that stores sales information is named `SALES.ACME.COM`.

```
CREATE PUBLIC DATABASE LINK sales.acme.com
  USING 'dbstring';
```

The example above creates a public database link to the `SALES` database. Notice the following points:

- A complete database name (both name and network domain) is specified for the database link name.
- The `CONNECT TO` clause is omitted, so ORACLE uses the local user/password to create a session in the remote database with the database link.
- The `USING` parameter of the statement specifies a database string *dbstring*:

```
CREATE DATABASE LINK sales
  CONNECT TO scott IDENTIFIED BY tiger
```

The second example creates a private database link for the creator of the link. Notice the following points:

- A partial database name (only the name component) is specified for the database link name. As a result, ORACLE automatically expands the database link name by appending the network domain of the local database to the specified database name to form a global database name.
- A `CONNECT TO` clause is explicitly specified. The specified remote user/password is used when the database link is used to establish a session in the remote database.
- A `USING` clause is omitted. Therefore, when the private database link is used, a public or network database link of the same name must be present and must indicate the database

string to use when establishing a connection to the remote database.

In all cases, the complete global name of a database link is accessible in the local data dictionary, in `DBA_DB_LINKS`, `ALL_DB_LINKS`, and `USER_DB_LINKS`. A database link's name must be unique in the scope in which it is created.



Trusted

Note: If you are using Trusted ORACLE in OS MAC mode, see the *Trusted ORACLE7 Server Administrator's Guide* for additional information about creating database links.

Privileges Required to Create a Database Link

The creator of a database link must have either the `CREATE DATABASE LINK` system privilege (to create private database links) or `CREATE PUBLIC DATABASE LINK` system privilege (to create public database links).

The remote accounts to which a database link points must exist in the remote database, and must have at least the `CREATE SESSION` system privilege. If a database link uses individual accounts in the remote database, the local user of the database link must have an account in the remote database. The database administrator of each database must specify privileges for each account. The requirements discussed in this paragraph are not checked when a database link is created; they are checked when a connection is attempted using the database link.

Controlling Connections Established by the RECO Background Process

In its attempt to recover an in-doubt distributed transaction, the RECO background process establishes connections to remote nodes involved in the transaction. To control the costs of connecting to other databases, you might want to control the length of time that such connections are held open on behalf of RECO using the `DISTRIBUTED_RECOVERY_CONNECTION_HOLD_TIME` initialization parameter.

Dropping Database Links

A database link is dropped using the `DROP DATABASE LINK` command. For example, to drop the public database link named `SALES`, enter the following statement:

```
DROP PUBLIC DATABASE LINK sales.acme.com;
```

Partial global database names are expanded using the network domain component of the local database.

Note: A database link should not be dropped if it is required to resolve an in-doubt distributed transaction. See “Troubleshooting Distributed Transaction problems” on page 15-15 for more information about in-doubt distributed transactions.

Privileges Required to Drop a Database Link

If the database link is private, only the owner can drop the database link. If the database link is public, you must have the DROP PUBLIC DATABASE LINK system privilege.

Examining Available Database Links

The data dictionary of each database stores the definitions of all the database links in that database. The USER/ALL/DBA_DB_LINKS data dictionary views show the database links that have been defined.

For example, assume that the local database’s global name is MKTG.ACME.COM. Also assume that the following CREATE DATABASE LINK statements have been issued by the same user:

```
CREATE DATABASE LINK hq.acme.com
CONNECT TO guest IDENTIFIED BY password

CREATE DATABASE LINK sales USING 'dbstring';
```

The following query lists all of the private database links contained in the schema associated with the current user issuing the query:

```
SELECT db_link, username, host
FROM user_db_links;
```

For example, if the user that owns the previously created database links (HQ and SALES) issues the query above, results similar to those below are returned:

DB_LINK	USERNAME	HOST
-----	-----	-----
HQ.ACME.COM	GUEST	
SALES.ACME.COM		dbstring

Notice that the USERNAME and HOST fields can be null if database link definitions do not indicate complete paths to the remote database.

Limiting the Number of Active Database Links

You can limit the number of connections from a user process to remote databases with the parameter OPEN_LINKS. This parameter controls the number of remote connections that any single user process can use concurrently, within a single SQL statement. To improve application performance, increase the value of this parameter if users need to access more databases at the same time; this allows the user to access all the remote data he needs without waiting for the local instance to close and open connections. For more information, see Appendix A.

Implementing Location Transparency

Location transparency means that the physical location of table data is not specified by (and may not be known by) the end-users and application developers of a database system. Location transparency is beneficial because remote data access is simplified, both local and remote object identities can be hidden from users, and objects can be moved with no effect on users or current applications.

Location transparency is not automatic, but it can be provided to end-users and applications when desired using several different methods:

- A locally defined view can provide location transparency by referencing remote tables in its definition.
- A local synonym can hide the database link used to access a remote table or view.
- The statements of a local procedure can access remote data and remote procedures. Users of the procedure simply call the procedure to access remote data.

Consider these issues when using views, synonyms, and procedures to provide location transparency.

Object Resolution and Location Transparency

When defining views, synonyms, and procedures that reference remote objects, consider the following issues so that your views, synonyms, and procedures always access the intended objects with the intended privileges:

- Unless all remote objects are explicitly qualified with their schema names, the accessed object is dependent on the connection established via the specified database link. For example, if you want a local synonym's remote base table to be SCOTT.EMP, explicitly specify schema SCOTT in the synonym definition:

```
CREATE SYNONYM emp FOR scott.emp@hq.acme.com;
```
- Database links defined with a CONNECT TO clause always connect to the remote database as the user specified in the CONNECT TO clause. Views, synonyms, and procedures that use this type of database link are always subject to the privileges and schema resolution of the specified remote user. Alternatively, when database links without a CONNECT TO clause are used in the definition of a view, synonym, or procedure, the remote connection is dependent on the local

user. Views and procedures should typically use database links with explicit accounts to ensure that they can function with the proper remote authorizations.

Circular References and Location Transparency

When views, procedures, and synonyms are used to implement location transparency, circular references are not detected by ORACLE, but can be controlled using resource limits (profiles), and specifically the SESSIONS limit. For example, assume that a public synonym in the SALES database, EMP, resolves to the public synonym EMP in the HQ database, and the public synonym EMP in the HQ database references the public synonym EMP in the SALES database. A SQL statement referencing the public synonym EMP in either SALES or HQ would terminate when the maximum number of sessions is exceeded.

Balancing Location Transparency and Security

The choice of using a view, synonym, or procedure for location transparency determines the degree to which the local and remote administrators are responsible for object security. The following example statements and sections outline the issues to consider when choosing among the options for location transparency.

Statement issued at remote database:

```
GRANT SELECT, DELETE ON scott.emp TO user1;
```

Statements issued at local database:

```
CREATE DATABASE LINK hr.acme.com
  CONNECT TO user1 IDENTIFIED BY password
  USING 'db_string';
CREATE VIEW admin.emp_view AS
  SELECT * FROM scott.emp@hr.acme.com;
CREATE PROCEDURE admin.fire_emp (enum NUMBER) AS
BEGIN
  DELETE FROM scott.emp@hr.acme.com
    WHERE empno = enum;
END;
CREATE SYNONYM admin.emp_syn FOR scott.emp@hr.acme.com;
```

Privilege Management With Views

Assume a local view (ADMIN.EMP_VIEW) references a remote table or view. The owner of the local view can grant only the object privileges on his view that have been granted the remote user referenced in the database link; this is similar to privilege management for views that reference local data. Therefore, local privilege management is possible when views are used for location transparency. For example, the user ADMIN can successfully grant the SELECT and DELETE privileges, but not the INSERT and UPDATE privileges for EMP_VIEW.

Views are a good choice for location transparency if unlimited local object privilege management is a requirement. For example, assume the local security administrator needs to selectively grant object privileges for several remote tables. The remote administrator can create a powerful user account that is granted many privileges for many remote tables. Then, the local administrator can create a private database link that connects to the powerful remote account and, in the same schema, create views to “mirror” the remote tables. The local administrator controls local privilege management for the remote tables by *granting* privileges on the local views. Also note that in this example, a private database link can be used by many users.

Privilege Management With Procedures

Assume a local procedure includes a statement that references a remote table or view (see example on previous page). The owner of the local procedure can grant the EXECUTE privilege to any user, thereby giving that user the ability to execute the procedure and access remote data.

In general, procedures aid in security; users that call a procedure can only perform the controlled operations of the procedure, and privileges for objects referenced within a procedure do not need to be explicitly granted to the calling users. Much like views, procedures are a good choice for location transparency if unlimited local privilege management is a requirement. For example, assume the local security administrator needs to selectively allow users to query and update several remote tables. The remote administrator can create a powerful user account granted many object privileges. Then, the local administrator can create a private database link that connects to the powerful remote account and, in the same schema, create procedures to query and modify the remote tables, as desired. The local administrator can control how local users can access the remote tables by selectively granting the EXECUTE privilege for the local procedures, thus controlling local privilege management for remote objects.

Privilege Management With Synonyms

Assume a local synonym is an alias for a remote object. The owner of the local synonym **cannot** grant any object privileges on the synonym to any other local user. This behavior is different from privilege management for synonyms that are aliases for local tables or views; in the case where a synonym is an alias for a remote object, local privileges for the synonym cannot be granted because this would amount to granting privileges for the remote object, which is not allowed. Therefore, no local privilege management can be performed when synonyms are used for location transparency; security for the base object is controlled entirely at the remote node. For example, the user ADMIN cannot grant any object privileges for the EMP_SYN synonym.

Unlike a database link referenced in a view or procedure definition, a database link referenced in a synonym is resolved by first looking for a private link owned by the schema in effect at the time the reference to the synonym is parsed. Therefore, to ensure the desired object resolution, it is especially important to specify the underlying object's schema in the definition of a synonym.

Limiting the Number of Distributed Transactions Per Node

The parameter `DISTRIBUTED_TRANSACTIONS` controls the number of possible distributed transactions in which a given instance can concurrently participate, both as a client and a server. If this limit is reached and a subsequent user tries to issue a SQL statement referencing a remote database, the statement is rolled back and the following error message is returned:

```
ORA-2042: too many global transactions
```

For example, assume that `DISTRIBUTED_TRANSACTIONS` is set to 10 for a given instance. In this case, only ten sessions can concurrently be in the process of a distributed transaction; if an eleventh session attempts to issue a DML statement requiring distributed access, an error message is returned to the session and the statement is rolled back.

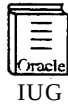
The database administrator should consider increasing the value of the initialization parameter `DISTRIBUTED_TRANSACTIONS` when an instance regularly participates in numerous distributed transactions and the above error message is frequently returned as a result of the current limit. Increasing the limit allows more users to concurrently issue distributed transactions.

If the `DISTRIBUTED_TRANSACTIONS` initialization parameter is set to zero, no distributed SQL statements can be issued in any session. Also, the RECO background process is not started at startup of the local instance and any in-doubt distributed transactions that may be present (from a previous network or system failure) cannot be automatically resolved by ORACLE; therefore, only set this initialization parameter to zero to prevent distributed transactions when a new instance is started and when it is certain that no in-doubt distributed transactions remained after the last instance shut down.

Setting the Commit Point Strength of an Instance

Set the commit point strength for each ORACLE database so that the most critical server can be “non-blocking” in the event of a failure during a two-phase commit. A database’s commit point strength should relate to the estimated number of collisions that can result from data locked by in-doubt transactions. For example, databases controlled by ORACLE on mainframe computers most likely have higher commit point strengths than databases controlled by minicomputers. In turn, databases on minicomputers most likely have higher commit point strengths than databases controlled by PC database servers. To determine such settings, it is necessary for all administrators of the distributed database system to communicate and establish proper guidelines for commit point strength settings.

A database instance’s commit point strength is set by the initialization parameter `COMMIT_POINT_STRENGTH`. The range of values is any integer from 0 to 255. For example, to set the commit point strength of a database to 200, include the following line in that database’s parameter file:



```
COMMIT_POINT_STRENGTH=200
```

The default value is operating system specific; see your installation or user’s guide for this value on your system.

Troubleshooting Distributed Transaction Problems

After a network or system failure occurs, it can cause the following types of problems for the users of the distributed database system:

- A two-phase commit being processed when a failure occurs may not be completed at all nodes of the session tree.
- If a failure persists (for example, if the network is down for a long period of time), the data exclusively locked by in-doubt transactions is unavailable to statements of other transactions.

The following sections describe these situations.

Failures that Interrupt a Two-Phase Commit

The user program that commits a distributed transaction is informed of a problem by one of the following error messages:

ORA-02050: transaction *ID* rolled back, some remote dbs may be in-doubt
ORA-02051: transaction *ID* committed, some remote dbs may be in-doubt
ORA-02054: transaction *ID* in-doubt

An application should save information about a transaction if it receives any of the above errors. This information can be used later if manual distributed transaction recovery is desired.

Note: The failure cases that prompt these error messages are beyond the scope of this book and are unnecessary to understand in order to administer the system.

No action is required by the administrator of any node that has one or more in-doubt distributed transactions due to a network or system failure. The automatic recovery features of ORACLE transparently complete any in-doubt transaction so that the same outcome occurs on all nodes of a session tree (that is, all commit or all rollback) once the network or system failure is resolved. However, in extended outages, the administrator may wish to force the commit or rollback of a transaction to release any locked data. Applications must account for such possibilities.

Failures that Prevent Access of Data

When a user issues a SQL statement, ORACLE attempts to lock the required resources to successfully execute the statement. However, if the requested data is currently being held by statements of other uncommitted transactions and continues to remain locked for an excessive amount of time, a time-out occurs. Consider the following two scenarios.

Example 1: Transaction Time-Out

A DML SQL statement that requires locks on a remote database may be blocked from doing so if another transaction (distributed or non-distributed) currently owns locks on the requested data. If these locks continue to block the requesting SQL statement, a time-out occurs, the statement is rolled back, and the following error message is returned to the user:

ORA-02049: time-out: distributed transaction waiting for lock

Since no data has been modified, no actions are necessary as a result of the time-out. Applications should proceed as if a deadlock has been encountered. The user who executed the statement can try to re-execute the statement later. If the lock persists, the user should contact an administrator to report the problem.

The timeout interval used in the above situation can be controlled using the initialization parameter `DISTRIBUTED_LOCK_TIMEOUT`. This

interval is in seconds. For example, to set the time-out interval for an instance to 30 seconds, the following line should be included in the associated parameter file:

```
DISTRIBUTED_LOCK_TIMEOUT=30
```

With the above time-out interval, the time-out errors discussed in the previous section occur if a transaction cannot proceed after 30 seconds of waiting for unavailable resources.

For more information about initialization parameters and editing parameter files, see Appendix A.

**Example 2:
Lock From In-Doubt
Transaction**

A query or DML statement that requires locks on a local database may be blocked from doing so indefinitely due to the locked resources of an in-doubt distributed transaction. In this case, the following error message is immediately returned to the user:

```
ORA-01591: lock held by in-doubt distributed transaction ID
```

In this case, the SQL statement is rolled back immediately. The user who executed the statement can try to re-execute the statement at a later time. If the lock persists, the user should contact an administrator to report the problem, including the ID of the in-doubt distributed transaction.

The chances of the above situations occurring are very rare, considering the low probability of failures during the critical portions of a two-phase commit. Even if such a failure occurs and assuming quick recovery from a network or system failure, problems are automatically resolved without manual intervention, and thus are likely to be resolved before they can be detected by users or database administrators.

**Manually Overriding
In-Doubt Transactions**

A database administrator can manually force the COMMIT or ROLLBACK of a local in-doubt distributed transaction. However, a specific in-doubt transaction should be manually overridden only when the following situations exist:

- The in-doubt transaction is locking data that is required by other transactions; this happens if users complain that the ORA-01591 error message is interfering with their transactions.
- An in-doubt transaction is preventing the extents of a rollback segment to be used by other transactions. The first portion of an in-doubt distributed transaction's local transaction ID corresponds to the ID of the rollback segment's ID, as listed by

the data dictionary views DBA_2PC_PENDING and DBA_ROLLBACK_SEGS.

- The failure that did not allow the two-phase commit to complete will not be corrected in an acceptable time period. Examples of such cases might include a telecommunication network that has been damaged or a damaged database that needs a substantial amount of time to complete recovery.

Normally, a decision to locally force an in-doubt distributed transaction should be made in consultation with administrators at other locations. Understand that a wrong decision can lead to database inconsistencies that must be manually corrected, which can be difficult to trace.

If the conditions above do not apply, ***always*** allow the automatic recovery features of ORACLE to complete the transaction. However, if any of the above criteria are met, the administrator should consider a local override of the in-doubt transaction. If a decision is made to locally force the transaction to complete, the database administrator should analyze available information with the following goals in mind:

- Try to find a node that has either committed or rolled back the transaction. If you can find a node that has already resolved the transaction, you can follow the action taken at that node.
- See if any information is given in the TRAN_COMMENT column of DBA_2PC_PENDING for the distributed transaction. Comments are included in the COMMENT parameter of the COMMIT command. For example, an in-doubt distributed transaction's comment might indicate the origin of the transaction and what type of transaction it is:

```
COMMIT COMMENT 'Finance/Accts_pay/Trans_type 10B';
```
- See if any information is given in the ADVICE column of DBA_2PC_PENDING for the distributed transaction. An application can prescribe advice about whether to force the commit or force the rollback of separate parts of a distributed transaction using the ADVISE parameter of the SQL command ALTER SESSION. The advice sent during the prepare phase to each node is the advice in effect at the time the most recent DML statement was executed at that database in the current transaction. For example, consider a distributed transaction that is moving an employee record from the EMP table at one node to the EMP table at another node. The transaction could protect the record, even when administrators independently force the in-doubt transaction at each node, by including the following sequence of SQL statements:


```

ALTER SESSION ADVISE COMMIT;
INSERT INTO emp@hq . . . ;      /* advice to commit at HQ */

ALTER SESSION ADVISE ROLLBACK;
DELETE FROM emp@sales . . . ; /* advice to roll back at SALES */

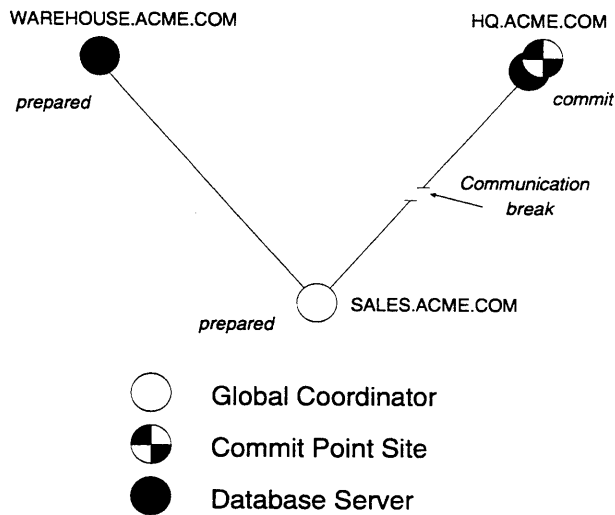
ALTER SESSION ADVISE NOTHING;

```

If the advice is followed to manually force the in-doubt transaction, the worst that can happen is that each node has a copy of the employee record being moved; the record cannot disappear.

Manual Override Example The following example shows a failure during the commit of a distributed transaction and how to go about gaining information before manually forcing the commit or rollback of the local portion of an in-doubt distributed transaction. Figure 15-1 illustrates the example.

FIGURE 15-1
An Example of an In-Doubt Distributed Transaction



In this failure case, the Prepare phase completed. However, during the Commit phase, the commit point site's commit message (the message telling the global coordinator that the transaction was committed at the commit point site) never made it back to the global coordinator, even though the commit point site committed the transaction. The italicized text indicates the state of all nodes involved in the in-doubt transaction.

You are the WAREHOUSE database administrator. The inventory data locked because of the in-doubt transaction is critical to other transactions, but cannot be accessed because the locks must be held until the in-doubt transaction either commits or rolls back.

Furthermore, you understand that the communication link between sales and headquarters cannot be resolved immediately. Therefore, you decide to manually force the local portion of the in-doubt transaction using the following steps:

1. Record user feedback.
2. Query the local DBA_2PC_PENDING view to obtain the global transaction ID and get other information about the in-doubt transaction.
3. Query the local DBA_2PC_NEIGHBORS view to begin tracing the session tree so that you can find a node that resolved the in-doubt transaction.
4. Check the mixed outcome flag after normal communication is re-established.

The following sections explain each step in detail for this particular example.

Record User Feedback The users of the local database system that conflict with the locks of the in-doubt transaction get the following error message:

```
ORA-01591: lock held by in-doubt distributed transaction 1.21.17
```

Here, 1.21.17 is the local transaction ID of the in-doubt distributed transaction in this example. The local database administrator should request and record this ID number from the users that report problems to identify in-doubt transactions that should be forced.

Query DBA_2PC_PENDING Query the local DBA_2PC_PENDING to gain information about the in-doubt transaction:

```
SELECT * FROM sys.dba_2pc_pending
WHERE local_trax_id = '1.21.17';
```

For example, when the previous query is issued at WAREHOUSE, the following information is returned. (Here, query results are transposed for clarity.)

Column Name	Value
LOCAL_TRAN_ID	1.21.17
GLOBAL_TRAN_ID	SALES.ACME.COM.55dlc563.1.93.29
STATE	prepared
MIXED	no
ADVICE	
TRAN_COMMENT	Sales/New Order/Trans_type 10B
FAIL_TIME	31-MAY-91
FORCE_TIME	
RETRY_TIME	31-MAY-91
OS_USER	SWILLIAMS
OS_TERMINAL	TWA139:
HOST	system1
DB_USER	SWILLIAMS
COMMIT#	

The global transaction ID is the common transaction ID that is the same on every node for a distributed transaction. It is of the form:

`global_database_name.hhhhhhhh.local_transaction_id`

Here, *global_database_name* is the database name of the global coordinator (where the transaction originates), *hhhhhhhh* is an internal database ID at the global coordinator (8 hexadecimal digits), and *local_tran_id* is the corresponding local transaction ID assigned on the global coordinator. Therefore, the last portion of the global transaction ID and the local transaction ID will match at the global coordinator; in the example, you can tell that WAREHOUSE is not the global coordinator because these numbers do not match.

The state of the transaction on this node is “prepared.” Therefore, WAREHOUSE is waiting for its coordinator to send either a commit or rollback message.

The transaction’s comment or advice may include information about this transaction. If so, use this comment to your advantage. In this example, the origin (the sales order entry application) and transaction type is included in the transaction’s comment. This information may reveal something that would help you decide whether to commit or rollback the local portion of the transaction.

If useful comments do not accompany an in-doubt transaction, you must complete some extra administrative work to trace the session tree and find a node that has resolved the transaction. Proceed to Step 3.

Query DBA_2PC_NEIGHBORS The purpose of this step is to climb the session tree such that you find coordinators, eventually reaching the global coordinator. Along the way, you might find a coordinator

that has resolved the transaction. If not, you can eventually work your way to the commit point site, which will always have resolved the in-doubt transaction.

The DBA_2PC_NEIGHBORS view provides information about connections associated with an in-doubt transaction. Information for each connection is different, based on whether the connection is inbound or outbound:

- If the connection is inbound, your node is subordinate (a server of) another node. In this case, the DATABASE column lists the name of the client database that connected to your node, and the DBUSER_OWNER column lists the local account used for the database link connection that corresponds to the in-doubt transaction.
- If the connection is outbound, your node is a client of other servers. In this case, the DATABASE column lists the name of the database link used to connect to the remote node, and the DBUSER_OWNER column lists the owner of the database link used within the in-doubt transaction.

Additionally, the INTERFACE column tells whether the local node or a subordinate node is the commit point site.

To trace the session tree, you can query the local DBA_2PC_NEIGHBORS view. In this case, you query this view on the WAREHOUSE database. (Here, query results are transposed for clarity.)

```
SELECT * FROM sys.dba_2pc_neighbors
       WHERE local_tran_id = '1.21.17'
       ORDER BY sess#, in_out;
```

Column Name	Value
LOCAL_TRAN_ID	1.21.17
IN_OUT	in
DATABASE	SALES.ACME.COM
DBUSER_OWNER	SWILLIAMS
INTERFACE	N
DBID	000003F4
SESS#	1
BRANCH	0100

The columns of particular interest in this view are the IN_OUT, DATABASE, DBUSER_OWNER, and INTERFACE columns. In this example, the returned information reveals that this node (WAREHOUSE) is a server of the node SALES, and that the connection

to WAREHOUSE was established via a database link that connected to the SWILLIAMS account.

At this point, you can contact the administrator at the located nodes and ask them to repeat Steps 2 and 3, using the global transaction ID.

Note: If you can directly connect to these nodes using another network, you can repeat Steps 2 and 3 yourself.

For example, the following results are returned when Steps 2 and 3 are performed at SALES and HQ, respectively:

At SALES: `SELECT * FROM sys.dba_2pc_pending
 WHERE global_tran_id = 'SALES.ACME.COM.55dlc563.1.93.29';`

Column Name	Value
LOCAL_TRAN_ID	1.93.29
GLOBAL_TRAN_ID	SALES.ACME.COM.55dlc563.1.93.29
STATE	prepared
MIXED	no
ADVICE	
TRAN_COMMENT	Sales/New Order/Trans_type 10B
FAIL_TIME	31-MAY-91
FORCE_TIME	
RETRY_TIME	31-MAY-91
OS_USER	SWILLIAMS
OS_TERMINAL	TWA139:
HOST	system1
DB_USER	SWILLIAMS
COMMIT#	

```
SELECT * FROM dba_2pc_neighbors  
      WHERE global_tran_id = 'SALES.ACME.COM.55dlc563.1.93.29'  
      ORDER BY sess#, in_out;
```

At SALES, there are three rows for this transaction (one for the connection to WAREHOUSE, one for the connection to HQ, and one for the connection established by the user). Information corresponding to the rows for the SALES and HQ connections is listed below:

Column Name	Value
LOCAL_TRAN_ID	1.93.29
IN_OUT	OUT
DATABASE	WAREHOUSE.ACME.COM
DBUSER_OWNER	SWILLIAMS
INTERFACE	N
DBID	55dlc563
SESS#	1
BRANCH	1
LOCAL_TRAN_ID	1.93.29
IN_OUT	OUT
DATABASE	HQ.ACME.COM
DBUSER_OWNER	ALLEN
INTERFACE	C
DBID	00000390
SESS#	1
BRANCH	1

The information from the previous query reveals several facts:

- SALES is the global coordinator because the local transaction ID and global transaction ID match. Also, notice that two outbound connections are established from this node, but no inbound links (this node is not a server of another node).
- HQ or one of its servers (none in this example) is the commit point site.

At HQ:

```
SELECT * FROM dba_2pc_pending
WHERE global_tran_id = 'SALES.ACME.COM.55dlc563.1.93.29';
```

Column Name	Value
LOCAL_TRAN_ID	1.45.13
GLOBAL_TRAN_ID	SALES.ACME.COM.55dlc563.1.93.29
STATE	COMMIT
MIXED	NO
ACTION	
TRAN_COMMENT	Sales/New Order/Trans_type 10B
FAIL_TIME	31-MAY-91
FORCE_TIME	
RETRY_TIME	31-MAY-91
OS_USER	SWILLIAMS
OS_TERMINAL	TWA139:
HOST	SYSTEM1
DB_USER	SWILLIAMS
COMMIT#	129314

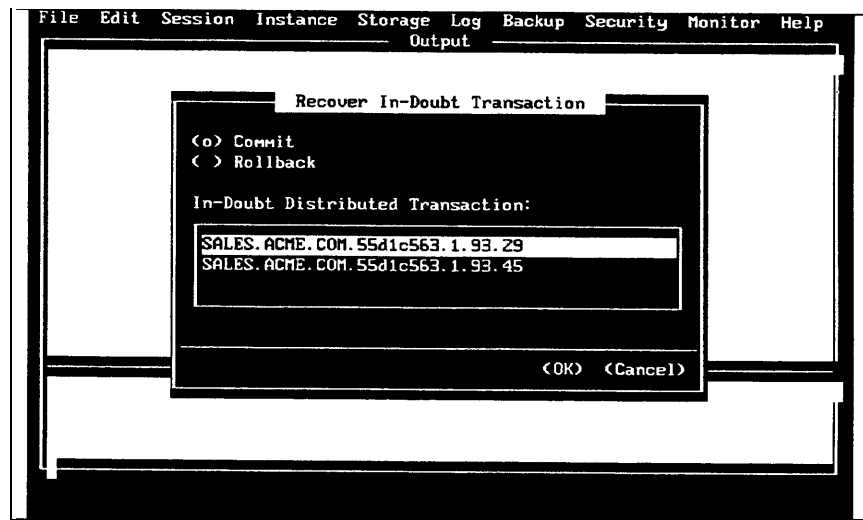
At this point, you have found a node that resolved the transaction: it has been committed. Therefore, you can force the in-doubt transaction to commit at your local database (see the following section for information on manually committing or rolling back in-doubt transactions). It is a good idea to contact any other administrators you know that could also benefit from your investigation.

Check for Mixed Outcome After you manually force a transaction to commit or rollback, the corresponding row in the pending transaction table remains. The STATE of the transaction is changed to “forced commit” or “forced abort,” depending on how you forced the transaction. Furthermore, once connections between the instances can be resumed, RECO checks the global outcome of the transaction. The MIXED column is changed to “yes” and the row for the transaction is not deleted if you forced the transaction the wrong way. If you ever see a transaction forced the wrong way, you should be aware that some global data inconsistency may exist. Eventually, you can purge unnecessary rows from the pending transaction table.

Manually Committing In-Doubt Transactions

The local database administrator can manually force an in-doubt transaction to commit using either the Recover In-Doubt Transaction dialog box of SQL*DBA or the SQL command COMMIT WORK with the FORCE option and a text string indicating either the local or global transaction ID of the in-doubt transaction to commit. Figure 15-2 shows the Recover In-Doubt Transaction dialog box.

FIGURE 15-2
The Recover In-Doubt
Transaction Dialog



The following statement is the command equivalent of the Recover In-Doubt Transaction dialog box in Figure 15-2:

```
COMMIT FORCE 'SALES.ACME.COM.55d1c563.1.93.29';
```

In both examples, the transaction is committed on the local node and the local pending transaction table records a value of “forced commit” for the STATE column of this transaction’s row.

Optionally, when forcing a transaction to commit, the SCN for the transaction can be specified; this feature allows you to commit an in-doubt transaction with the SCN assigned when it was committed at other nodes, thus maintaining the synchronized commit time of the distributed transaction even in the event of failure. An SCN should only be specified when you can determine the SCN of the same transaction already committed at another node.

For example, assume you want to manually commit a transaction with the global transaction ID *global_id*. First, query the DBA_2PC_PENDING view of a remote database also involved with the transaction in question and note the SCN used for the commit of the transaction at that node. Specify the SCN (a decimal number) when committing the transaction at the local node. For example, assume the noted SCN is 829381993:

```
COMMIT FORCE 'global_id', 829381993;
```

Manually Rolling Back In-Doubt Transactions

The local database administrator can manually force an in-doubt transaction to rollback using either the Recover In-Doubt Transaction dialog box of SQL*DBA or the SQL command ROLLBACK WORK command with the FORCE option and a text string indicating either the local or global transaction ID of the in-doubt transaction to rollback. For example, to rollback the in-doubt transaction with the local transaction ID of 2.9.4, enter the following statement:

```
ROLLBACK FORCE '2.9.4';
```

Once this statement is executed, the transaction is rolled back on the local node and the local pending transaction table records a value of “forced abort” for the STATE column of this transaction’s row.

Note: An in-doubt distributed transaction cannot be forced to rollback to a savepoint; only the entire in-doubt distributed transaction can be forced to rollback.

Privileges Required to Manually Commit or Rollback In-Doubt Transactions

To manually force the commit or rollback of an in-doubt transaction issued by yourself, you must have been granted the FORCE TRANSACTION system privilege. To force the commit or rollback of another user’s distributed transaction, you must have the FORCE ANY

TRANSACTION system privilege. Both privileges can be obtained either explicitly or via a role.

Note: Forcing the commit or rollback of an in-doubt distributed transaction does not affect the status of the operator's current transaction.

Changing Connection Hold Time

After a distributed transaction fails, the connection from the local site to the remote site does not necessarily close immediately; instead, it remains open in case communication can be restored quickly, without having to re-establish the connection. You can set the length of time that the connection remains open with the database parameter `DISTRIBUTED_RECOVERY_CONNECTION_HOLD_TIME`. A high value minimizes the cost of reconnecting after failures, but causes the local database to consume more resources. In contrast, a lower value minimizes the cost of resources kept locked during a failure, but increases the cost of reconnecting after failures. The default value of the parameter is 200 seconds. See Appendix A for more information.

Setting a Limit on Distributed Transactions

The database parameter `DISTRIBUTED_TRANSACTIONS` sets a maximum on the number of distributed transactions in which a database can participate. You should increase the value of this parameter if your database is part of many distributed transactions; the default value is operating system specific.

In contrast, if your site is experiencing an abnormally high number of network failures, you can temporarily decrease the value of this parameter. Doing so limits the number of in-doubt transactions in which your site takes part, and thereby limits the amount of locked data at your site, and the number of in-doubt transactions you might have to resolve.

For more information on this parameter, see Appendix A.

Testing Distributed Transaction Recovery Features

If you like, you can force the failure of a distributed transaction to observe RECO automatically resolving the local portion of the transaction. Alternatively, you might be interested in forcing a distributed transaction to fail so that you can practice manually resolving in-doubt distributed transactions and observing the results. The following sections describe the features available and the steps necessary to perform such operations.

Forcing a Distributed Transaction to Fail

Comments can be included in the `COMMENT` parameter of the `COMMIT` statement. To intentionally induce a failure during the

two-phase commit of a distributed transaction, include the following comment in the COMMENT parameter:

```
COMMIT COMMENT 'ORA-2PC-CRASH-TEST-n';
```

where *n* is one of the following integers:

<i>n</i>	<i>Effect</i>
1	Crash commit point site after collect
2	Crash non-commit point site after collect
3	Crash before prepare (non-commit point site)
4	Crash after prepare (non-commit point site)
5	Crash commit point site before commit
6	Crash commit point site after commit
7	Crash non-commit point site before commit
8	Crash non-commit point site after commit
9	Crash commit point site before forget
10	Crash non-commit point site before forget

For example, the following statement returns the following messages if the local commit point strength is greater than the remote commit point strength and both nodes are updated:

```
COMMIT COMMENT 'ORA-2PC-CRASH-TEST-7';
```

```
ORA-02054: transaction #.##.## in-doubt
```

```
ORA-02059: ORA-CRASH-TEST-n in commit comment
```

At this point, the in-doubt distributed transaction appears in the DBA_2PC_PENDING view. If enabled, RECO automatically resolves the transaction rather quickly.

Privileges Required to Induce Two-Phase Commit Failures You can induce two-phase commit failures via the previous comments only if both the local and remote sessions have the FORCE ANY TRANSACTION system privilege. Otherwise, an error is returned if you attempt to issue a COMMIT statement with a crash comment.

Disabling and Enabling RECO

The recoverer background process, RECO, can be enabled and disabled using the ALTER SYSTEM command with the ENABLE/DISABLE DISTRIBUTED RECOVERY options, respectively. For example, you might want to temporarily disable RECO so that you can force the failure of a two-phase commit and manually resolve the in-doubt transaction. The following statement disables RECO:

```
ALTER SYSTEM DISABLE DISTRIBUTED RECOVERY;
```

Alternatively, the following statement enables RECO so that in-doubt transactions are automatically resolved:

```
ALTER SYSTEM ENABLE DISTRIBUTED RECOVERY;
```

Note: Single-process instances (for example, a PC running MS-DOS have no separate background processes, and therefore no RECO process. Therefore, when a single-process instance that participates in a distributed database is first started, distributed recovery must be manually enabled using the statement above. See your installation or user's guide for more information about distributed transaction recovery for single-process instances.



MANAGING TABLE SNAPSHOTS

This chapter explains how to manage table snapshots (local copies of a remote table). The topics in this chapter include:

- creating, altering, and dropping snapshots
- refreshing snapshots
- troubleshooting snapshot problems
- creating, altering, managing, and dropping snapshot logs
- duplicating tables manually

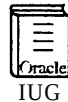
Except for manual table duplication, the information in this chapter applies only to systems using ORACLE with both the distributed option and the procedural option.

Managing Snapshots

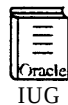
A snapshot is a read-only copy of a table or of data from multiple tables. A snapshot is periodically refreshed to reflect a recent, transaction-consistent state of the tables that it replicates.

The following sections explain how to create, alter, and drop snapshots.

Preparing to Create Snapshots



Before anyone can create any snapshots or snapshot logs in a database, you must run the CATSNAP.SQL script on both the database that contains the master table and the database that will contain snapshots; you must be connected as SYS. The name and location of the CATSNAP.SQL script are operating system specific; see your installation or user's guide. This script is usually run during database creation of sites with the procedural option.



In addition, the procedures described in this chapter for administering snapshots require that you run the script DBMSSNAP.SQL, which creates packages of procedures for refreshing snapshots. You must run this script, while connected as SYS, on both the master table's database and the snapshot's database. The name and location of the DBMSSNAP.SQL script are operating system specific; see your installation or user's guide. This script is also usually run by default during database creation for sites with the procedural option.



Also, if you want snapshots to be refreshed automatically, you can, depending on your operating system, start a periodic batch job or write a utility to connect periodically to the database containing the snapshots and call the procedure DBMS_SNAPSHOT.REFRESH_ALL(), which refreshes the snapshots. On many platforms, Oracle provides a utility program like this; you specify when you start the utility the name of a file that indicates which user (and password) the utility should use in connecting, and the interval at which it should connect. The exact name, location, and operation of the refresh utility are operating system dependent; in many cases, the utility is called REFSNAP. See your installation or user's guide for information about this utility on your platform.

Creating Snapshots

You create a local snapshot using the SQL command CREATE SNAPSHOT. As when creating tables, you can specify storage characteristics, extent sizes and allocation, and the tablespace to hold the snapshot, or a cluster to hold the snapshot (in which case all of the previous options do not apply). You can also specify how the snapshot is to be refreshed and the distributed query that defines the snapshot; this is unique to snapshots. For example, the following CREATE

SNAPSHOT statement defines a local snapshot to replicate the remote EMP table located in NY:

```
CREATE SNAPSHOT emp_sf
  PCTFREE 5 PCTUSED 60
  TABLESPACE users
  STORAGE (INITIAL 50K NEXT 50K PCTINCREASE 50)
  REFRESH FAST
    START WITH sysdate
    NEXT sysdate + 7
  AS SELECT * FROM emp@ny;
```

Whenever a snapshot is created, it is immediately filled with the rows returned by the query that defines the snapshot. Thereafter, the snapshot is refreshed as specified by the REFRESH clause; see “Refreshing Snapshots” on page 16-5.

Naming Snapshots

Snapshots are contained in a user’s schema. A snapshot’s name must be unique with respect to other objects in the schema. Keep snapshot names to 23 or fewer characters. If a snapshot name contains more than 23 characters, ORACLE automatically (and uniquely) truncates the prefixed names of the underlying table and views, so that they comply with the naming rules for schema objects.

Creating a Clustered Snapshot

You can create a snapshot in a cluster, just as you can a table. For example, the following statement creates a snapshot named EMP_DALLAS in the EMP_DEPT cluster:

```
CREATE SNAPSHOT emp_dallas
...
CLUSTER emp_dept
... ;
```

The storage parameters of the cluster’s data segment are used for the storage of the clustered snapshot, even if storage parameters are specified for the snapshot.

Privileges Required to Create Snapshots

To create a snapshot, the following sets of privileges must be granted:

- To create a snapshot in your own schema, you must have the CREATE SNAPSHOT, CREATE TABLE, CREATE VIEW, and CREATE INDEX (only for simple snapshots) system privileges, as well as SELECT privilege on the master tables.
- To create a snapshot in another user’s schema, you must have the CREATE ANY SNAPSHOT, CREATE ANY TABLE, CREATE ANY VIEW, and CREATE ANY INDEX (only for simple snapshots) system privileges, as well as SELECT privilege on the master table. Additionally, the owner of the

snapshot must have the appropriate SELECT privilege on the master tables.

In either case, the owner of the snapshot must also have sufficient quota on the tablespace intended to hold the snapshot.

The large set of privileges required to create a snapshot is due to the underlying objects that must also be created on behalf of the snapshot.

Understanding the Internals of Snapshot Creation

When a snapshot is created, several operations are performed internally by ORACLE:

- ORACLE (at the snapshot node) creates a table to store the rows retrieved by the snapshot's defining query; this is the snapshot's *base table*. The table's name is SNAP\$_*snapshotname*. (This base table of a snapshot should not be altered in anyway; do not add triggers or integrity constraints to a snapshot's base table or directly modify data in a snapshot's base table.)
- ORACLE creates a read-only view on the SNAP\$ table for queries issued against the snapshot. This view uses the name provided by the CREATE SNAPSHOT statement. (No DML operations are allowed on this view because it is read-only.)
- ORACLE creates a second local view on the remote master table. It uses this view when it refreshes the snapshot. This view is named MVIEW\$_*snapshotname*.

Additionally, if the snapshot is a simple snapshot, ORACLE creates an index on the SNAP\$ table. This index is named PK\$_*snapshotname*.

All of these internal objects are created in the schema of the snapshot. Do not alter, change data in, or delete these objects manually.

Setting Storage Parameters for Snapshots

How you should set storage options for a snapshot depends on the type of snapshot (simple or complex):

- In general, a simple snapshot's storage options should mimic the storage options for its master table, since they share the same characteristics. If a number of master tables are clustered in the master database, you should probably cluster the corresponding snapshots in the remote database.

Note: If a simple snapshot does not duplicate all columns of its master table, modify the snapshot storage items accordingly.

- Because a complex snapshot is always completely refreshed, set its PCTFREE to 0 and PCTUSED to 100 for maximum efficiency.

You can change a snapshot's storage parameters using the ALTER SNAPSHOT command. For example, the following command alters the EMP snapshot's PCTFREE parameter

```
ALTER SNAPSHOT emp PCTFREE 10;
```

You cannot change a snapshot's defining query; you must first drop the snapshot and then re-create it.

Privileges Required to
Alter a Snapshot

To alter a snapshot's storage parameters, the snapshot must be contained in your schema or you must have the ALTER ANY SNAPSHOT and ALTER ANY TABLE system privileges.

Refreshing Snapshots

Each snapshot is individually refreshed in a separate transaction to make the snapshot reflect a more recent state of its master tables. There are two types of refreshes: a fast refresh and a complete refresh. A *fast refresh* uses the snapshot log of a master table to refresh a simple snapshot (only simple snapshots can execute a fast refresh). A *complete refresh* entirely replaces the existing data in a simple or complex snapshot. Also, snapshots can be refreshed automatically or manually.

Consider the following issues when deciding how to refresh a snapshot:

- In most cases with a simple snapshot, use fast refreshes because they are more efficient than complete refreshes.
- If the master tables receive a consistent number of updates in a given time period, automatically refresh dependent snapshots.
- After bulk loads to the master tables, manually refresh all snapshots based on the master table. This propagates the new rows of the master tables to associated snapshots.

Configuring
Automatic Refreshes

To configure snapshots to be automatically refreshed, perform the following steps:

1. Create or alter each snapshot to be refreshed automatically, specifying automatic refresh options.
2. Start the refresh utility program on the database server that has the snapshots (not the master table).

To configure a specific snapshot to automatically refresh, specify a refresh option in the REFRESH clause of a CREATE SNAPSHOT or ALTER SNAPSHOT statement. The refresh options available are FAST, COMPLETE, and FORCE. FORCE, the default, performs a fast refresh if possible, or a complete refresh otherwise. The START WITH parameter specifies the time for a snapshot's initial automatic refresh; if omitted, the initial automatic refresh of the snapshot is evaluated using the

creation time and the NEXT parameter. The NEXT parameter specifies the automatic refresh interval for the snapshot. NEXT is always evaluated from the last refresh, automatic or manual.

The automatic refresh settings for a snapshot are stored in the data dictionary. Refresh settings in the data dictionary are updated if they are altered via an ALTER SNAPSHOT statement and every time the snapshot is refreshed (automatically or manually). When setting a snapshot's refresh settings, understand the following behavior

- The dates or date expressions in the START WITH and NEXT parameters must evaluate to a future point in time; if they do not, an error message is returned. A date literal must be enclosed in single quotes, while date expressions do not require quotes (see example below).
- If a snapshot should be refreshed periodically at a set interval, use the NEXT parameter with a date expression similar to "SYSDATE + 7". For example, if you set the automatic refresh interval to "SYSDATE + 7" on Monday and manually refresh the snapshot on Thursday, "SYSDATE + 7" now evaluates to every Thursday, not Monday. Alternatively, if you always want to automatically refresh the snapshot at a specific time, regardless of the last refresh (for example, every Monday), the NEXT parameter should specify a date expression similar to "NEXT_DAY(TRUNC(SYSDATE), 'MONDAY')".

Table 16-1 lists some common date expressions used for snapshot refresh intervals.

TABLE 16-1
Examples of Common
Refresh Intervals

Date Expression	Evaluation
SYSDATE + 7	Exactly seven days from the last refresh
NEXT_DAY(TRUNC(SYSDATE), 'MONDAY') + 15/24	Every Monday at 3PM
NEXT_DAY(ADD_MONTHS(TRUNC(SYSDATE,'Q'),3), 'THURSDAY')	The first Thursday of each quarter

The following examples of the REFRESH clause show valid combinations and date expressions:

```
CREATE SNAPSHOT snap
. . .
REFRESH COMPLETE
      START WITH '01_SEP_92'
      NEXT sysdate + 7
AS . . . ;

ALTER SNAPSHOT snap REFRESH FAST;
```

```
ALTER SNAPSHOT snap
  REFRESH COMPLETE
  NEXT next_day(trunc(sysdate), 'MONDAY') + 15/24;
```

The first statement creates the SNAP snapshot, specifying complete automatic refreshes, the first of which occurs on September 1st, 1992, with an automatic refresh interval of seven days from the most recent refresh (whether automatic or manual). The second statement specifies that automatic refreshes should be fast. The third statement specifies that automatic refreshes should be complete, and changes the automatic refresh interval to be every Monday at 3PM.

For automatic refreshes to take place, the refresh utility program, supplied with ORACLE, runs on the database server. This utility notifies ORACLE about the automatic refresh of any appropriate snapshot. When the refresh utility is started, an interval can be specified that indicates how often the utility “wakes up” to check for snapshots due to be automatically refreshed. The exact name and location of the refresh utility program are operating system dependent.

Troubleshooting Automatic Refresh Problems Several factors can prevent the automatic refresh of snapshots: not executing the refresh utility, an intervening network or instance failure, or an instance shutdown. When any of these factors prevents the automatic refresh of a snapshot, the snapshot remains “due” to be refreshed. Snapshots due to be refreshed are automatically refreshed once the refresh utility is executed and any network failure is resolved.

Manually Refreshing Snapshots

You can refresh a snapshot manually, whether or not it is also automatically refreshed by ORACLE. You refresh an individual snapshot using the REFRESH stored procedure of the package DBMS_SNAPSHOT, with the following syntax:

```
dbms_snapshot.refresh ('[schema.]name' [, 'refresh_option'])
```

The argument *schema.name* is the name of the snapshot to refresh. The optional argument *refresh_option* specifies the type of refresh to perform; 'F' or 'f' indicates a fast refresh, 'C' or 'c' indicates a complete refresh, and '?' indicates the default refresh option of the snapshot.

For example, the following EXECUTE statement within SQL*DBA causes a fast refresh of the EMP snapshot:

```
EXECUTE dbms_snapshot.refresh ('emp', 'f');
```

You can refresh all snapshots due to be automatically refreshed (those not yet refreshed by the refresh utility) by calling the REFRESH_ALL stored procedure of the DBMS_SNAPSHOT package. This procedure

refreshes each snapshot due to be automatically refreshed, according to its automatic refresh settings:

```
EXECUTE dbms_snapshot.refresh_all;
```

The REFRESH_ALL procedure accepts no arguments. Snapshots not configured for automatic refresh are not affected by the REFRESH_ALL stored procedure. A single execution of the REFRESH_ALL procedure refreshes all snapshots due to be automatically refreshed. As this procedure executes, each snapshot due to be automatically refreshed is individually refreshed in a separate transaction. If a system failure interrupts the procedure, some snapshots may have been refreshed while others are still due to be refreshed. If this happens, execute the REFRESH_ALL procedure again.

Privileges Required to Manually Refresh Snapshots To alter the refresh settings for a snapshot or to manually refresh the snapshot, you must own the snapshot or have the ALTER ANY SNAPSHOT system privilege. If you are manually refreshing a snapshot using the fast method, you must also have access to the snapshot log on the master table: you must either own the snapshot log or have SELECT privilege on it. (The name of a snapshot log is MLOG\$_master_table_name, where master_table_name is the name of the master table. See “Managing Snapshot Logs” on page 16-10 for more information.)

Any user with the ALTER ANY SNAPSHOT system privilege can execute the REFRESH_ALL procedure to manually refresh any snapshots due to be automatically refreshed.

Refreshing Snapshots and Referential Integrity

If referential integrity is enforced on one master table or between multiple master tables, refresh related snapshots so that the referential integrity of the master tables is propagated to the associated snapshots.

For example, consider the EMP and DEPT master tables, related via referential integrity on the DEPTNO column. Referential integrity of EMP and DEPT is propagated to the corresponding snapshots EMP_SNAP and DEPT_SNAP only if you can refresh EMP_SNAP and DEPT_SNAP without any modifications taking place on the EMP and the DEPT table between the individual refreshes.

To refresh a single snapshot with a master table that has a self-referencing integrity constraint, use the following steps:

1. Lock, in exclusive mode, the master table with the self-referential integrity constraint:

```
LOCK TABLE emp IN EXCLUSIVE MODE;
```

2. Refresh the snapshot using the REFRESH procedure of the DBMS_SNAPSHOT package. For example, when using SQL*DBA, enter the following statement:

```
EXECUTE dbms_snapshot.refresh('emp', '?');
```

To refresh snapshots of master tables related by referential integrity, use the following procedure:

1. Start a session with an interactive tool (such as SQL*Plus or SQL*DBA) and establish a connection with a database in the distributed database system.
2. Issue a LOCK TABLE statement to exclusively lock the tables related via referential integrity:

```
LOCK TABLE emp, dept IN EXCLUSIVE MODE;
```

3. Start a second session with an interactive tool and establish a connection to the database that contains the snapshots to be refreshed. Note that the first session should remain active to preserve the table locks established in Step 2.
4. Refresh the snapshots that correspond to the tables locked in Step 2. For example, the following statements refresh the EMP and DEPT snapshots:

```
EXECUTE dbms_snapshot.refresh ('emp', '?');
```

```
EXECUTE dbms_snapshot.refresh ('dept', '?');
```

Declarative Constraints and Snapshots

Do not create declarative constraints on snapshots or snapshot logs.

Indexing Snapshots

To increase the query performance when using a snapshot, you can create indexes for the snapshot. To index a column (or columns) of a snapshot, you must create the index on the underlying "SNAP\$_" table created to hold the rows of the snapshot. Do not use declarative constraints to create an index; instead, use the CREATE INDEX statement.

Operations on a Master Table that Affect Snapshots

All changes made by INSERT, UPDATE, and DELETE statements issued against a table are reflected in associated snapshots when the snapshots are refreshed.

However, changes made by a TRUNCATE command on master tables are not propagated automatically to simple snapshots; they are

propagated to complex snapshots. If you truncate a master table, completely refresh any associated simple snapshots.

If you drop a master table, any associated snapshots remain and continue to be accessible. An associated snapshot log (if present) of a dropped master table also remains. However, when you attempt to refresh a snapshot based on a non-existent master table, ORACLE returns an error.

Snapshots and Media Failure

As the result of a media failure, either a database that contains a master table of a snapshot or a database with a snapshot may need to be recovered. If you have a snapshot or master table in a database that you must recover, see “Recover Databases with Snapshots” on page 19-5.

Dropping Snapshots

You can drop a snapshot independently of its master tables or the snapshot log. To drop a local snapshot; use the SQL command DROP SNAPSHOT. For example:

```
DROP SNAPSHOT emp;
```

If you drop the only snapshot of a master table, you should also drop the snapshot log of the master table, if there is one.

Privileges Required to Drop a Snapshot

Only the owner of a snapshot or a user with the DROP ANY SNAPSHOT, DROP ANY TABLE, and DROP ANY VIEW system privileges can drop a snapshot.

Managing Snapshot Logs

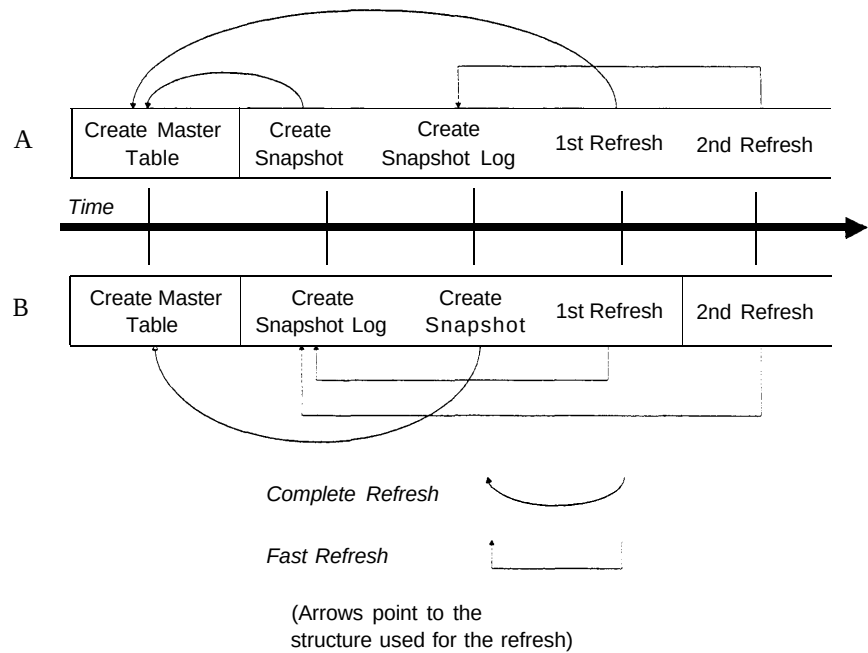
A snapshot log is a table, in the same database as the master table for a snapshot, that is associated with the master table. Its rows list changes that have been made to the master table, and information on which snapshots have and have not been updated to reflect those changes. It can be used by simple snapshots to perform a fast refresh; it cannot be used in complex snapshots.

These sections explain how to create, manage, and drop snapshot logs.

Creation Order of a Simple Snapshot and the Snapshot Log

If you are creating a simple snapshot, it is more efficient to create the snapshot log before the snapshot. Figure 16-1 illustrates the two orders of creation.

FIGURE 16-1
Creation Order of a Simple
Snapshot and the Snapshot Log



In Method A, the first refresh of the snapshot cannot use the log because the log cannot reflect all updates entered between the creation of the snapshot and the creation of the snapshot log; therefore, two complete refreshes are necessary.

In contrast, Method B only requires one complete refresh (when creating the snapshot); subsequent refreshes can immediately use the snapshot log. If the master table is large or a number of simple snapshots are based on the same master table, creating the snapshot log before the snapshots can be much more efficient.

Creating a Snapshot Log

You can create a snapshot log to decrease the amount of processing and time needed to refresh the simple snapshot. Snapshot logs cannot be used with complex snapshots. A snapshot log is associated with a single master table; likewise, a master table can have only one snapshot log. If multiple simple snapshots are based on the same master table, they all use the same snapshot log.

Create a snapshot log in the same database as the master table using the SQL command `CREATE SNAPSHOT LOG`. You can set storage options for the snapshot log's data blocks, extent sizes and allocation, and

tablespace to hold the snapshot log. For example, the following statement creates a snapshot log associated with the EMP table:

```
CREATE SNAPSHOT LOG ON emp
  TABLESPACE users
  STORAGE (INITIAL 10K NEXT 10K PCTINCREASE 50)
  PCTFREE 5;
```

Naming Snapshot Logs

ORACLE automatically creates the snapshot log in the schema that contains the master table. Since you cannot specify a name for the snapshot log (one is implicitly given by ORACLE), uniqueness is not a concern.

The Internals of Snapshot Log Creation

When you create a snapshot log, ORACLE performs several operations internally:

- ORACLE creates a table to store the ROWID and timestamp of rows updated in the master table. The table is named *MLOG\$_master_table_name*.
- ORACLE creates an AFTER ROW trigger on the master table to insert the ROWIDs and timestamps of inserted, updated, and deleted rows into the snapshot log. The trigger is named *TLOG\$_master_table_name*.

Note: Because snapshot log creation implicitly creates an AFTER ROW trigger on the master table, you cannot explicitly create any other AFTER ROW triggers for a master table.

The underlying table for a snapshot log and associated trigger are contained in the same schema as the master table.

Do not alter or change data in these objects.

Privileges Required to Create Snapshot Logs

If you own the master table, you can create an associated snapshot log if you have the CREATE TABLE and CREATE TRIGGER system privileges. If you are creating a snapshot log for a table in another user's schema, you must have the CREATE ANY TABLE and CREATE ANY TRIGGER system privileges. In either case, the owner of the snapshot log must have sufficient quota in the tablespace intended to hold the snapshot log.

The privileges required to create a snapshot log directly relate to the privileges necessary to create the underlying objects associated with a snapshot log.

Setting Storage Parameters for Snapshot Logs

Set a snapshot log's storage options as follows:

- Set PCTFREE to 0, and PCTUSED to 100.
- Set extent storage parameters according to the update activity (number of INSERT, UPDATE, and DELETE statements) on the master table.

Each row in a snapshot log takes approximately 13 bytes (six bytes for a ROWID and seven bytes for a timestamp). Use this number to calculate how much space a snapshot log requires, using the procedure described in “Calculating Space Required by Non-Clustered Tables” on page 8-17.

Altering Snapshot Log Storage Parameters

You can alter a snapshot log's storage parameters using the SQL command ALTER SNAPSHOT LOG. For example:

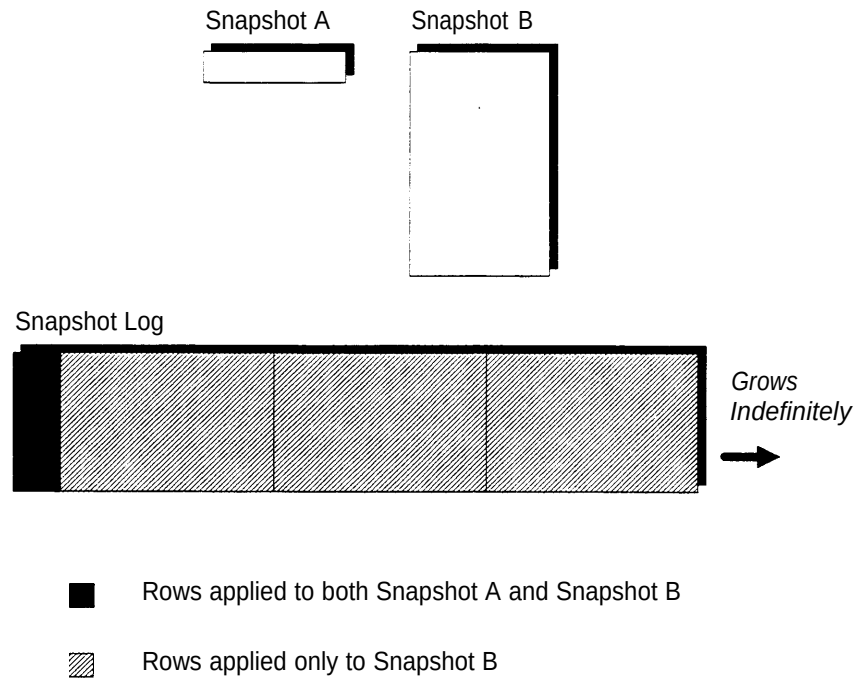
```
ALTER SNAPSHOT LOG sale_price
  PCTFREE 25
  PCTUSED 40;
```

Privileges Required to Alter Snapshot Logs Only the owner of the master table, or a user with the ALTER ANY TABLE and ALTER ANY TRIGGER system privileges can alter the storage parameters of a snapshot log.

Managing Snapshot Log Space Use

ORACLE automatically tracks which rows in a snapshot log have been used during the refreshes of snapshots, and purges these rows from the log so that the log does not grow endlessly. Because multiple simple snapshots can use the same snapshot log, rows used in the refresh of one snapshot may still be needed to refresh another snapshot; ORACLE does not delete rows from the log until all snapshots have used them. However, this automated feature can cause a snapshot log to grow indefinitely if a single associated snapshot is never refreshed. Figure 16-2 illustrates this case.

FIGURE 16-2
Managing a Snapshot Log



Snapshot B is regularly refreshed. However, ORACLE cannot purge the rows used during the refresh of Snapshot B because Snapshot A needs them for its next refresh. This situation occurs when you have several simple snapshots based on the same master table and either:

- One snapshot is not configured to be automatically refreshed by ORACLE; it has to be manually refreshed.
- One snapshot has a large refresh interval, such as every year.
- A network failure has prevented an automatic refresh of one or more of the snapshots based on the master table.

Keep the snapshot log as small as possible to minimize the space it uses. To reduce the number of rows in a snapshot log, you can either refresh the snapshots associated with the log, or shrink the log by deleting the rows only required by the Nth least recently refreshed snapshots. To do the latter, execute the PURGE_LOG stored procedure of the DBMS_SNAPSHOT package, with the following syntax:

```
dbms_snapshot.purge_log ('table', purge_number [, 'DELETE'])
```

The argument *table* is the name of the master table. The argument *purge_number* indicates the number of least recently refreshed snapshots whose rows you want to remove from snapshot log. For

example, the following statement deletes rows needed to refresh the two least recently refreshed snapshots:

```
EXECUTE dbms_snapshot.purge_log('master_table', 2);
```

To delete all rows in the snapshot log, indicate a high number of snapshots to disregard, as in this example:

```
EXECUTE dbms_snapshot.purge_log('master_table', 9999);
```

This statement completely purges the snapshot log that corresponds to MASTER_TABLE, if fewer than 9999 snapshots are based on MASTER_TABLE.

Note: A simple snapshot whose rows have been purged from the snapshot log must be completely refreshed the next time it is refreshed.

The optional argument DELETE guarantees that rows are deleted from the snapshot log for at least one snapshot, and can override the setting for the argument purge_number. For example, the following statement deletes rows from the least recently refreshed snapshot that actually has dependent rows in the snapshot log:

```
EXECUTE dbms_snapshot.purge_log('master_table', 0, 'DELETE');
```

Reducing Space Allocation for a Snapshot Log

If a snapshot log grows and allocates many extents, purging the log of rows does not reduce the amount of space allocated for the log. To reduce the space allocated for a snapshot log, either truncate or drop and re-create the snapshot log. The first refresh of each simple snapshot subsequent to truncating the log must be a complete refresh. Fast refreshes can then resume.

Privileges Required to Delete Rows From a Snapshot Log The owner of a snapshot log or a user with the DELETE ANY TABLE system privilege can purge rows from the snapshot log by executing the PURGE_LOG procedure.

Dropping Snapshot Logs

You can drop a snapshot log independently of its master table or any existing snapshots. You might decide to drop a snapshot log if one of the following is true:

- All simple snapshots of a master table have been dropped.
- All simple snapshots of a master table are to be completely refreshed, not fast refreshed.

To drop a local snapshot log, use the SQL command DROP SNAPSHOT LOG, as in:

```
DROP SNAPSHOT LOG emp_log;
```

Privileges Required to
Drop a Snapshot Log

Only the owner of the master table, or a user with the DROP ANY
TABLE system privilege can drop a snapshot log.

Manually Copying Tables Among Nodes of a Distributed Database

As an alternative to creating snapshots, you can manually replicate tables among the nodes of a distributed database. Manual table replication is often useful if the master table experiences high update activity while read-only replicas only need to periodically reflect the changes to the master table.

To create the local replica of the remote master table use a CREATE TABLE statement:

```
CREATE TABLE local_replica  
AS SELECT * FROM remote_master;
```

You can grant local privileges on the replica, create associated indexes and views, and so on.

When you want to update a local replica of the remote master table, drop and re-create the local replica. However, note the following issues:

- When you drop a replica, you also drop any associated indexes, triggers, and integrity constraints. Therefore, you must re-create indexes, triggers, and integrity constraints.
- When you drop a local replica, all object privileges granted to users or roles on the local replica are revoked. After re-creating the replica, regrant object privileges to users and roles.
- When you drop a local replica, all views based on the replica continue to exist, but become invalid. After you create the replica, all dependent views become valid if the new version of the replica has the same structure as the previous replica.

All privileges granted on a dependent view are preserved throughout the operation of updating a local replica. Therefore, consider creating a view of a local table replica and granting privileges on this view rather than on the replica itself, thereby avoiding regranteeing privileges on the replica.

PART

VI

DATABASE BACKUP AND RECOVERY

17

ARCHIVING REDO INFORMATION

This chapter explains how to create and maintain the archived redo log. Topics include:

- deciding whether to archive
- specifying the archiving file name format and destination
- turning archiving on and off
- tuning archiving
- checking the archiving status



Parallel

If you are using ORACLE with the Parallel Server, see the *ORACLE7 Parallel Server Administrator's Guide* for additional information about archiving in the environment.

Choosing Between NOARCHIVELOG and ARCHIVELOG Mode

Consider these issues when deciding whether to use archiving:

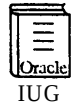
- In the event of a database failure, including disk failure, you can recover all committed transactions of the database if you are using ARCHIVELOG mode. If you are using NOARCHIVELOG mode, however, you can only **restore** (not recover) the database to the point of the most recent full database backup. You cannot recover subsequent transactions.
- If the database is operating in ARCHIVELOG mode, the entire database can be open and available for normal use while you back up or recover all or part of the database.

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; only full backups taken while the database is closed can be used to restore a database operating in NOARCHIVELOG mode. Therefore, if you decide to operate a database in NOARCHIVELOG mode, take full database backups at regular, frequent intervals.

- If **all** databases in a distributed database operate in ARCHIVELOG mode, you can perform coordinated distributed database recovery. However, if **any** database in a distributed database uses NOARCHIVELOG mode, recovery of a global distributed database (to make all databases consistent) is limited by the last full backup of any database operating in NOARCHIVELOG mode.
- Extra administrative operations are required to manage the files of the archived redo log.
- You must have a dedicated tape drive or additional disk space to store the archived redo log files if the database operates in ARCHIVELOG mode.

If a database operates in ARCHIVELOG mode, you must also decide how filled groups of the online redo log are to be archived; an instance can be configured to have ORACLE automatically archive filled online redo log files, or you can manually archive filled groups.

Turning Archiving On and Off



You set a database's initial archiving mode as part of database creation. In most cases, you can use the default of NOARCHIVELOG mode at database creation because there is no need to archive the redo information generated then. Once the database has been created, decide whether to change from the initial archiving mode.



Note: If a database is automatically created during ORACLE installation, the initial archiving mode of the database is operating system specific. See your installation or user's guide.

Once a database has been created, you can switch the database's archiving mode on demand. However, you should generally not switch the database between archiving modes.

Setting the Initial Database Archiving Mode at Database Creation

When you create the database, you set the initial archiving mode of the redo log in the CREATE DATABASE statement. If you do not specify either ARCHIVELOG or NOARCHIVELOG, NOARCHIVELOG is the default. See Chapter 2 for more information about creating a database.

Changing the Database Archiving Mode After Database Creation

To switch a database's archiving mode between NOARCHIVELOG and ARCHIVELOG mode, use the SQL command ALTER DATABASE with the ARCHIVELOG or NOARCHIVELOG option. For example, 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 the database's archiving mode can be switched. Archiving cannot be disabled if any data files need media recovery.

2. Back up the database.

Before making any major alteration to a database, always backup the database to protect against any problems that might occur. For more information about database backup, see Chapter 18.

3. Perform any operating system specific steps (*optional*).



If you want to archive filled groups, you may have to execute some additional steps at this point, depending on your operating system; see your installation or user's guide for details for your system. These steps may involve exiting SQL*DBA to configure how ORACLE will perform the archiving of the filled groups. Once this operation is complete, start SQL*DBA again and continue to Step 4.

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

To enable or disable archiving, the database must be mounted but not open. For more information about starting an instance and mounting a database, see Chapter 3.

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. See the *ORACLE7 Parallel Server Administrator's Guide* for more information about switching the archiving mode when using the ORACLE Parallel Server.



5. 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 or not to enable ORACLE to archive groups of online redo log files automatically as they fill. See the following sections for more information about the automatic and manual archiving options.

Enabling Automatic Archiving



If your operating system permits, you can enable automatic archiving of the online redo log. Under this option, you need take no action to copy a group after it fills; ORACLE automatically archives groups after they are filled. For this convenience alone, automatic archiving is the method of choice for archiving the filled groups of online redo log files. See your installation or user's guide to determine whether or not this is a valid option for your ORACLE Server.

If automatic archiving is enabled, manual archiving is still possible; see "Performing Manual Archiving" on page 17-6. Always specify an archived redo log destination and filename format when enabling automatic archiving; see "Specifying the Archived Redo Log Filename Format and Destination" on page 17-10.

Automatic archiving can be enabled before or after instance startup.

Enabling Automatic Archiving at Instance Startup

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

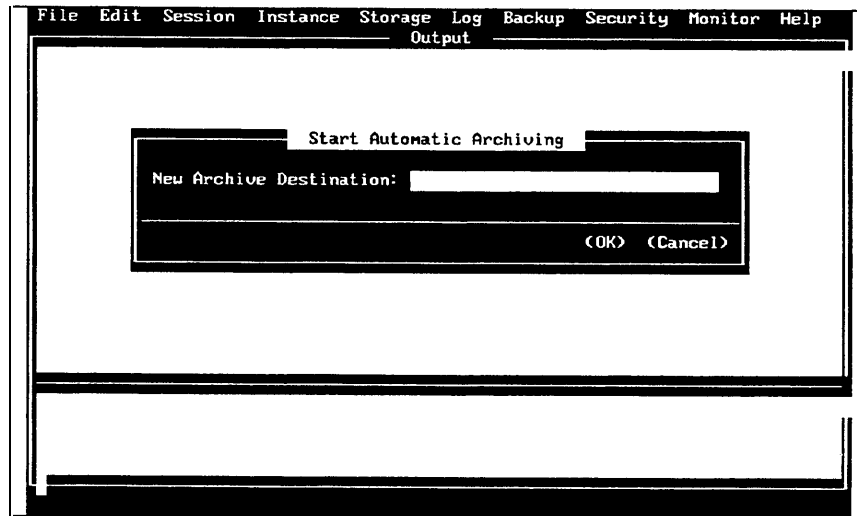
```
LOG_ARCHIVE_START=TRUE
```

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 Start Automatic Archiving dialog box of SQL*DBA or the SQL command ALTER SYSTEM with the ARCHIVE LOG START parameter; you can optionally include the archiving destination. Figure 17-1 shows the Start Automatic Archiving dialog box.

FIGURE 17-1
The Start Automatic Archiving Dialog



The following statement is the command equivalent of the Start Automatic Archiving dialog box in Figure 17-1:

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

Using either of the options above, the instance does not have to be shut down to enable automatic archiving; however, if an instance is shut down and restarted after automatic archiving is enabled, the instance is reinitialized using the settings of the parameter file, which may or may not enable automatic archiving.

Privileges Required to Enable Automatic Archiving After Instance Startup To enable automatic archiving after instance startup, you must be connected to ORACLE as INTERNAL. See “Connecting as INTERNAL” on page 1-4.

Disabling Automatic Archiving

You can disable automatic archiving of the online redo log groups at any time. However, once automatic archiving is disabled, you must manually archive groups of online redo log files in a timely fashion. If a database is operated in ARCHIVELOG mode, automatic archiving is disabled, and 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 the necessary archiving is performed.

Automatic archiving can be disabled at or after instance startup.

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 either the Stop Auto Archive option of the Logs menu of SQL*DBA or the SQL command ALTER SYSTEM with the ARCHIVE LOG STOP parameter. The following statement is the command equivalent of selecting the Stop Auto Archive menu option:

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

If ARCH is archiving a redo log group when you attempt to disable automatic archiving, ARCH 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; however, if an instance is shut down and restarted after automatic archiving is disabled, the instance is reinitialized using the settings of the parameter file, which may or may not enable automatic archiving.

Privileges Required to Disable Automatic Archiving After Instance Startup To disable automatic archiving after instance startup, you must be connected as INTERNAL and have the ALTER SYSTEM privilege.

Performing Manual Archiving

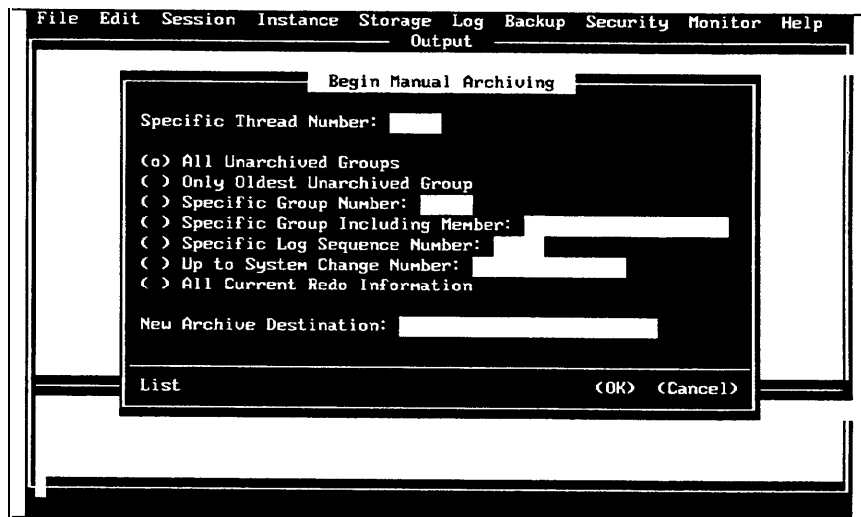
If a database is operating in ARCHIVELOG mode, inactive groups of filled online redo log files must be archived. You can manually archive

groups of the online redo log whether or not automatic archiving is enabled:

- If automatic archiving is not enabled, 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. (However, 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.)

Manually archive inactive groups of filled online redo log members using the Begin Manual Archive dialog box of SQL*DBA or the SQL command ALTER SYSTEM with the ARCHIVE LOG clause. Figure 17-2 shows the Begin Manual Archive dialog box.

FIGURE 17-2
The Begin Manual
Archive Dialog



The following statement is the command equivalent of the Begin Manual Archive dialog box in Figure 17-2:

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



With both manual or automatic archiving, you need to specify a thread only when you are using the ORACLE Parallel Server. See the *ORACLE7 Parallel Server Administrator's Guide* for more information.

Privileges Required to
Manually Archive an
Online Redo Log Group

To manually archive a filled online redo log group, you must be connected as INTERNAL. See “Connecting as INTERNAL” on page 1-4.

Tuning Archiving

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

For these large database sites, 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). The following sections describe how to adjust these parameters for either kind of tuning. The parameters themselves are described in more detail in Appendix A.

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 cause the archiver to 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. (This maximum is operating system dependent; see your installation or user's guide.) If the performance of the system drops significantly while the archiver is working, adjust LOG_ARCHIVE_BUFFER_SIZE lower, until system performance is no longer reduced when the archiver 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, consider creating additional redo log file groups. Adding groups can ensure that a group is always available for ORACLE to use. See Chapter 5 for information.

Increasing Archiving Speed

To increase archiving performance (for example, if you want to stream input to a tape drive), use multiple archive buffers, so that the archiver process can read the archive log at the same time that it writes the output log. You can set LOG_ARCHIVE_BUFFERS to 2, but for a very fast tape drive you might 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.

Displaying Archiving Status Information

Query the V\$DATABASE view to see the current archiving mode:

```
SELECT log_mode FROM sys.v$database;
```

```
LOG_MODE
-----
NOARCHIVELOG
```

The V\$ARCHIVE and V\$LOG data dictionary views also contain archiving information of a database. For example, the following query lists all log groups for the database and indicates the ones that remain to be archived:

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

```
GROUP#    ARC
-----
1      YES
2      NO
```

The List Archive Status option of the Logs menu in SQL*DBA and the command ARCHIVE LOG with the LIST parameter also show archiving information for the connected instance. For example:

```
ARCHIVE LOG LIST;
```

Database log mode	ARCHIVELOG
Automatic archival	ENABLED
Archive destination	<i>destination</i>

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 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) is *destination* (corresponds to LOG_ARCHIVE_DEST or an overriding destination).
- 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.

Privileges Required to List Archive Status

To list archive status information, you must be connected as INTERNAL. See “Connecting as INTERNAL” on page 1-4.

Specifying the Archived Redo Log Filename Format and Destination

When the database is used in ARCHIVELOG mode, ORACLE must know the archived redo log filename format and destination so that automatic or manual archiving creates uniquely named archived redo log files in the proper location.

Archived redo log files are uniquely named as specified by the LOG_ARCHIVE_FORMAT parameter. Filename format is operating system specific; for most operating systems it consists of a text string, one or more parameters, and a filename extension. When a filled online redo log group is archived, the archiving process concatenates the supplied text string with the return values of the specified parameters to create uniquely identified archived redo log files. Each parameter has an upper bound, which is operating system dependent. Table 17-1 lists the parameters that can be included in a filename format and corresponding examples to show how the parameter affects the filenames created by the archiving process.

TABLE 17-1
Archived Redo Log Filename
Format Parameters

Parameter	Description	Example*
%T	Thread number, left-zero-padded.	arch0000000001
%t	Thread number, not padded.	arch1
%S	Log sequence number, left-zero-padded.	arch0000000251
%s	Log sequence number, not padded.	arch251

* Assume LOG_ARCHIVE_FORMAT=arch%parameter, and the upper bound for all parameters is 10 characters.

The different options are provided so that you can customize the archived redo log filenames as you need. For example, you might want to take into account the operating system sorting algorithm used to list filenames.



The %T and %t are useful only when the ORACLE Parallel Server is used; see the *ORACLE7 Parallel Server Administrator's Guide* for more information about these filename format parameters and the term "thread."

In a non-Parallel Server configuration, you must decide whether to use %S or %s to identify each archived redo log file uniquely. The following is a typical example of a common archived redo log filename format:

```
LOG_ARCHIVE_FORMAT = arch%S.arc
```

Here, *arch* is the filename, %S is the zero-padded log sequence parameter, and *.arc* is the file extension. Assuming the upper bound for the %S parameter is four, this filename format generates archived redo log filenames of the following format:

```
arch0001.arc
arch0002.arc
arch0003.arc
.
.
```

Take into account the maximum operating system filename length when specifying the archive filename format; if ARCH or a user process attempts to archive a file and the supplied filename format is too large, the process fails to archive the file.

Note: If no archived filename format is specified using LOG_ARCHIVE_FORMAT, a default filename format is used by ORACLE. This default filename format is operating system specific.

The archived redo log destination is also operating system specific. For most operating systems, the archive redo log destination points to a disk drive and a file directory; if permitted by your ORACLE Server, this destination can also point to a tape drive dedicated to ORACLE for archiving filled online redo log files.

The archived redo log destination is determined at instance startup by the LOG_ARCHIVE_DEST initialization parameter, but can be overridden while the instance is up:

- If a database's parameter file is edited to include a destination using the LOG_ARCHIVE_DEST parameter, the current instance must be shut down and restarted to read the new parameter file.
- If the current instance cannot be shut down, but the archived redo log destination must be specified or changed for automatic archiving, use the Start Automatic Archiving dialog box of SQL*DBA or an ALTER SYSTEM ARCHIVE LOG START 'destination' statement to override the automatic archiving destination.
- During manual archiving, a specified destination overrides the default archived redo log destination. However, automatic archiving continues to use the current automatic archive destination. If no destination is specified, ORACLE automatically uses the destination specified by the LOG_ARCHIVE_DEST parameter of the parameter file used to start the instance. If no destination is supplied by the LOG_ARCHIVE_DEST parameter, a default destination is used. The default destination is operating system dependent.



See your installation or user's guide for more information about the LOG_ARCHIVE_FORMAT and LOG_ARCHIVE_DEST initialization parameters, and the default archived redo log filename format and destination.

BACKING UP A DATABASE

This chapter explains how to back up the data in an ORACLE database. Topics include:

- guidelines for backing up
- creating a backup strategy
- taking full and partial database backups
- taking a control file backup
- handling a failure during backup
- using the Export and Import utilities for backups

Guidelines for Database Backups

The following sections provide guidelines to consider in backing up an ORACLE database.

Plan Backup and Recovery Strategies Before Database Creation

Before you create an ORACLE database, you should decide how you plan to protect the database against potential disk failures, etc. If such planning is not considered before database creation, database recovery may not be possible if a disk failure damages the data files, online redo log files, or control files of a database. See “Creating a Backup Strategy” on page 18-4 if you are not familiar with typical backup strategies for a database.

Test Backup and Recovery Strategies



You should test your backup and recovery strategies in a test environment before and after you move to a production system. By testing your backup and recovery strategies, you can test the thoroughness of your strategies and minimize problems before they occur in a real situation. Performing test recoveries on a regular basis ensures that your archiving and backup procedures work. It also helps you stay familiar with recovery procedures, so that you are less likely to make a mistake in a crisis.

Perform Operating System Backups Frequently and Regularly

Frequent and regular full or partial database backups are essential for any recovery scheme. The frequency of backups should be based on the number of changes to database data (such as insertions, updates, and deletions of rows in existing tables, and addition of new tables) that are performed. If a database’s data is changed at a high rate, database backup frequency should be proportionally high. Alternatively, if a database is mainly read-only and few updates are issued, the database can be backed up less frequently.

Appropriately Backup the Database Before and After Modifying the Database Structure

If you make any of the following structural changes, perform an appropriate backup of your database immediately before and after completing the alteration:

- Create or drop a tablespace.
- Add or rename (relocate) a data file in an existing tablespace.
- Add, rename (relocate), or drop an online redo log group or member.

The appropriate type of backup depends on the archiving mode of the database, as described below:

- If a database is operated in ARCHIVELOG mode, only a control file backup (using the ALTER DATABASE command with the BACKUP CONTROLFILE option) is required before and after a structural alteration. (You can backup other parts of the database as well, though.)
- If the database is operated in NOARCHIVELOG mode, a full database backup should be taken immediately before and after the modification, including all data files, redo log files, and control files.

Back Up Often-Used Tablespaces Frequently

If a database is operated in ARCHIVELOG mode, it is acceptable to back up the data files of an individual tablespace or even a single data file. This option is useful if a portion of a database is used more extensively than others, such as the SYSTEM tablespace and tablespaces that contain rollback segments. By taking more frequent backups of the extensively used data files of a database, you gather more recent copies of the data files. As a result, if a disk failure damages the extensively used data files, the more recent backup can restore the damaged files and only a small number of changes to data need to be applied to roll the restored file forward to the time of the failure, thereby reducing database recovery time.

Keep Older Backups

How long you should keep an older database backup depends on the choices you want for database recovery. If you might want to recover to a past point-in-time, you need a database backup taken before that point-in-time. For a database operating in NOARCHIVELOG mode, this means a full database backup; for a database operating in ARCHIVELOG mode, this means a backup of each data file, taken individually or together, taken any time before the desired recovery point-in-time, and a backup of the associated control file that reflects the database's structure at the point-in-time of recovery.

For added protection, consider keeping two or more backups previous to the current backup, in case part of the current backup is lost or damaged.

Export Database Data for Added Protection and Flexibility

Because the ORACLE Export utility can selectively export specific objects, you might consider exporting portions or all of a database for supplemental protection and flexibility in a database's backup strategy. However, database exports are not a substitute for operating system backups and cannot provide the same complete recovery advantages that the built-in functionality of ORACLE offers.

Consider Distributed Database Backups

If a database is a node in a distributed database, consider the following guidelines:

- All databases in the distributed database system should be operated in the same archiving mode.
- If the databases in a distributed database system are operating in ARCHIVELOG mode, backups at each node can be performed autonomously (individually, without time coordination).
- If the databases in a distributed database system are operating in NOARCHIVELOG mode, full backups must be performed at the same (global) time, to plan for global distributed database recovery. For example, if a database in New York is backed up at midnight EST, the database in San Francisco should be backed up at 9PM PST. For more information about distributed database recovery when databases are operating in NOARCHIVELOG mode, see “Coordinate Distributed Recovery” on page 19-3.

Creating a Backup Strategy

Before you create an ORACLE database, decide how you plan to protect the database against potential failures. The two primary questions to ask before developing a backup strategy are as follows:

- **Is it acceptable to lose any data if a disk failure damages some of the files that constitute a database?** If it is not acceptable to lose any data in the event of a problem, the database must be operated in ARCHIVELOG mode, ideally with a mirrored online redo log. If it is acceptable to lose a limited amount of data in the event of a disk failure, you can operate the database in NOARCHIVELOG mode and avoid the extra work required to archive filled online redo log files.
- **Does the database need to be available at all times (that is, twenty-four hours per day, seven days per week)?** If so, it is not feasible to operate the database in NOARCHIVELOG mode because the required full database backups (taken while the database is shutdown) cannot be made frequently, if at all. Therefore, high-availability databases always operate in ARCHIVELOG mode to take advantage of online data file backups.

The following sections explain some common backup strategies based on the archiving mode of the database.

Backup Strategies in NOARCHIVELOG Mode

If a database is operated in NOARCHIVELOG mode, filled groups of online redo log files are not being archived. Therefore, the only protection against a disk failure is the most recent full backup of the database. The following list describes when to perform full backups for a database operated in NOARCHIVELOG mode:

- Plan to take full backups regularly, according to the amount of work that you can afford to lose. For example, if you can afford to lose the amount of work accomplished in one week, make a full backup once per week; if you can afford to lose only a day's work, make a full backup every day. For large databases with a high amount of activity, it is usually not acceptable to lose any work. Therefore, the database should be operated in ARCHIVELOG mode, and the appropriate backup strategies should be used.
- Any time that you alter the physical structure of a database operating in NOARCHIVELOG mode, immediately take a full database backup. An immediate full backup protects the new structure of the database not reflected in the previous full backup.

Backup Strategies in ARCHIVELOG Mode

If a database is operating in ARCHIVELOG mode, filled groups of online redo log files are being archived. Therefore, the archived redo log coupled with the online redo log and data file backups can protect the database from a disk failure, providing for complete recovery from a disk failure to the instant that the failure occurred. The following list describes a common backup strategy for a database operating in ARCHIVELOG mode:

- When the database is initially created, perform a full backup of the entire database. This initial full backup is the foundation of your backups because it provides copies of all data files and the control file of the associated database.



Note: When you perform this initial full backup, make sure that the database is in ARCHIVELOG mode first. Otherwise, the backed up database files will contain the NOARCHIVELOG mode setting.

- Subsequent full backups are not required, and if a database must remain open at all times, full backups are not feasible. Instead, you can take partial backups to update the backups of a database.
- Take online or offline data file backups to update backed up information for the database (supplementing the full, initial

backup). In particular, the data files of extensively used tablespaces should be backed up frequently to reduce database recovery time, should recovery ever be required. If a more recent data file backup is used to restore a damaged data file, fewer archived redo logs need to be applied to the restored data file to roll it forward to the time of the failure.

Whether you should take online or offline data file backups depends on the availability requirements of the data—online data file backups are the only choice if the data being backed up must always be available.

- Every time you make a structural change to the database, take a control file backup, using the ALTER DATABASE command with the BACKUP CONTROLFILE option.



Note: If the control file does not contain the name of a data file and you have no backup of that data file, you cannot recover the file if it is lost. Also, do **not** use operating system utilities to backup the control file in ARCHIVELOG mode, unless you are performing a full, offline backup.

- Usually online redo log files never need to be backed up when a database is operated in ARCHIVELOG mode; they are already archived. (However, you should backup online redo log files if you make a full, offline backup of the database.) A mirrored online redo log further safeguards against a single point of failure with respect to the online redo log.

Taking Backups

The following sections cover the procedures necessary to complete full and partial backups.

Listing Database Files Before Backup

Before taking a full or partial database backup, identify the files for backup. Obtain a list of data files for the database by querying the V\$DATAFILE view:

```
SELECT name FROM sys.v$datafile;
```

Obtain a list of online redo log files for a database using the query below:

```
SELECT name FROM sys.v$logfile;
```

These queries list the data files and online redo log files of a database, respectively, according to the information in the current control file of the database.

Obtain the names of the current control files of the database by issuing the following statement within SQL*DBA:

```
SHOW PARAMETER control_files;
```



Utilities

Note: For more information about the SHOW command of SQL*DBA, see the *ORACLE7 Server Utilities User's Guide*.

Whenever you take a control file backup, store the full list of data files and online redo log files with the corresponding control file backup; this way, the database's physical structure at the time the control file is backed up is clear and on record.

Performing Full Backups

Take a full backup of all files that constitute a database after the database is shut down to system-wide use in normal priority. **A full backup cannot be taken if the database is open or has been shut down because of an instance failure or abort;** in such cases, a backup is not a full backup because the files are not consistent with respect to the current point-in-time. (The files that constitute the database are the data files, online redo log files, and control file.)

Full backups do not require the database be operated in a specific archiving mode; a full backup can be taken if a database is operating in either ARCHIVELOG or NOARCHIVELOG mode.

The set of backup files that result from a full backup are consistent; all files correspond to the same point in time. If database recovery is necessary, these files can completely restore the database to an exact point in time. After restoring the backup files, additional recovery steps may be possible to restore the database to a more current time if the database is operated in ARCHIVELOG mode and online redo logs are not restored.



Note: A backup control file created during a full database backup should only be used with the other files taken in that backup, to restore the full backup. It should not be used for complete or incomplete database recovery. Unless you are taking a full database backup, you should back up your control file using the ALTER DATABASE command with the BACKUP CONTROLFILE option; see "Control File Backups" on page 18-15 for the steps involved.

Preparing to Take a Full Backup

To guarantee that a database's data files are consistent, always shut down the database in **normal** priority before making a full database

backup; never perform a full database backup after an instance failure or after the database is shut down with immediate or abort priority (that is, using a SHUTDOWN IMMEDIATE or SHUTDOWN ABORT statement). In both these cases, the data files are likely not to be consistent with respect to a specific point-in-time.

Steps for a Full Backup

To make a full backup, perform the following steps:

1. Shut down the database with normal priority.

To make a full backup, all files used by the database must be closed by shutting down the database. Do not make a full backup when the instance is aborted or stopped because of a failure; reopen the database and shut it down cleanly before making a full backup.

See Chapter 3 for more information about database shutdown.

2. Back up all files that constitute the database.

Use operating system commands or a backup utility to make backups of all data files, online redo log files, and a single control file of the database. If you are minoring the online redo log, back up all members of each group, since it is not guaranteed that any one member of a group is complete. Also backup the parameter files associated with the database.

Operating system backups can be performed:

- within SQL*DBA, using the HOST dialog box or HOST command
- outside of SQL*DBA, with the operating system commands or a backup utility



See your installation or user's guide for more information about making operating system backups of files.

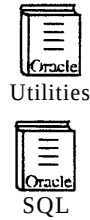
3. Restart the database.

Once you have finished backing up all data files, online redo log files, and a single control file of the database, you can restart the database. (See Chapter 3 for more information about database startup.)

Performing Partial Backups

The following sections explain the procedures necessary to complete the different types of partial backups:

- online tablespace and data file backups
- offline tablespace and data file backups



Partial backups should only be taken (and in some cases can only be taken) if a database is operating in ARCHIVELOG mode; partial backups cannot be used to restore a database operating in NOARCHIVELOG mode.

Note: For complete information about the dialog boxes of SQL*DBA and SQL commands ALTER TABLESPACE and ALTER DATABASE used in the following sections, see the *ORACLE7 Server Utilities User's Guide* and the *ORACLE7 Server SQL Language Reference Manual*, respectively.

Online Tablespace and Online Data File Backups

All data files of an individual online tablespace or specific data files of an online tablespace can be backed up while the tablespace and data files are currently online and in use for normal database operation. To perform an online backup of an entire tablespace, or a specific data file of an online tablespace, use the following steps:

1. Identify the data files.

If you are backing up a specific data file, use the fully specified filename of the data file.

Before beginning a backup on an entire tablespace, identify all of the tablespace's data files using the DBA_DATA_FILES data dictionary view. For example, assume that the USERS tablespace is to be backed up. To identify the USERS tablespace's data file, you can query the DBA_DATA_FILES view:

```
SELECT tablespace_name, file_name
       FROM sys.dba_data_files
       WHERE tablespace_name = 'USERS';
```

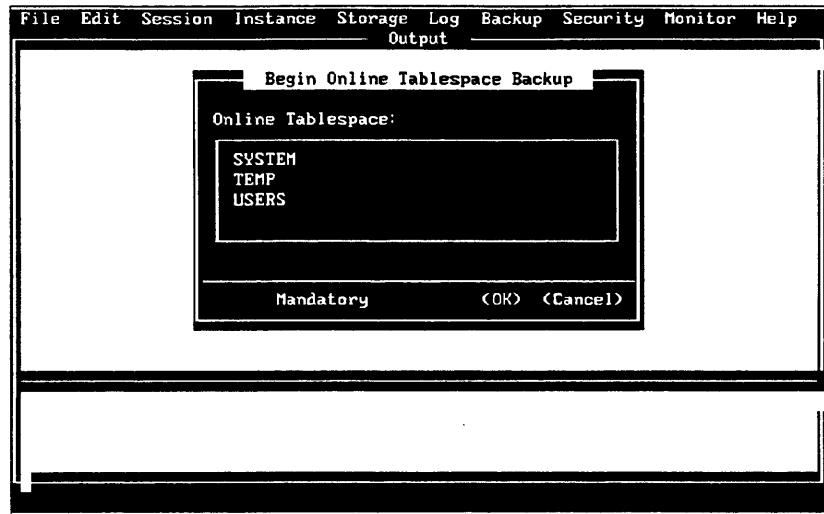
TABSPACE_NAME	FILE_NAME
USERS	filename1
USERS	filename2

Here, *filename1* and *filename2* are fully specified filenames corresponding to the data files of the USERS tablespace. See Appendix B for more information about the DBA_DATA_FILES data dictionary view.

2. Mark the beginning of the online tablespace backup.

To prepare the data files of an online tablespace for backup, use either the Begin Online Tablespace Backup dialog box of SQL*DBA or the SQL command ALTER TABLESPACE with the BEGIN BACKUP option. Figure 18-1 shows the Begin Online Tablespace Backup dialog.

FIGURE 18-1
The Begin Online Tablespace
Backup Dialog



The following statement is the command equivalent of the Begin Online Tablespace Backup dialog in Figure 18-1:

```
ALTER TABLESPACE users BEGIN BACKUP;
```

Note: If you forget to mark the beginning of an online tablespace backup (using either method) before backing up an online tablespace, the backup data files are not useful for subsequent recovery operations. Attempting to recover such a backup might return no errors, but can result in inconsistent data at a later date.

3. Back up the online data files.

At this point, you can backup the online data files of the online tablespace:

- from within SQL*DBA, using the HOST command
- by exiting SQL*DBA and entering the operating system commands or starting the backup utility

See the *ORACLE7 Server Utilities User's Guide* for information on see your installation or user's guide for more information about making operating system backups of files.

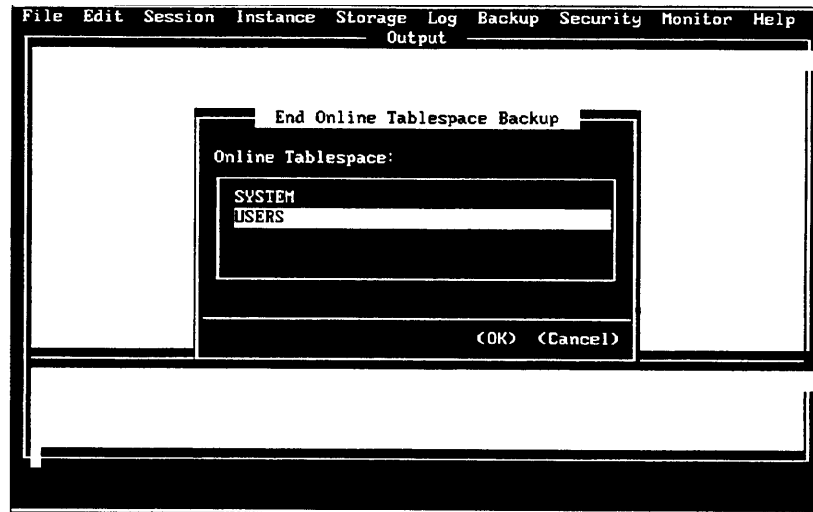
4. Mark the end of the online tablespace backup.

After the data files of the online tablespace have been backed up, indicate the end of the online backup using either the End Online Tablespace Backup dialog box of SQL*DBA or the SQL command ALTER TABLESPACE with the END BACKUP



option. Figure 18-2 shows the End Online Tablespace Backup dialog box.

FIGURE 18-2
The End Online Tablespace Backup
Dialog Box



The following statement is the command equivalent of the End Online Tablespace Backup dialog box in Figure 18-2:

```
ALTER TABLESPACE users END BACKUP;
```

Note: If you forget to indicate the end of an online tablespace backup and the database is subsequently shut down, ORACLE assumes that media recovery is necessary at the next instance start up. Recovery may require the use of the archived redo log.

Determining Data File Backup Status To see the backup status of a data file (whether it has been backed up), you can often use the data dictionary table V\$BACKUP. This table lists all online files and gives their backup status. It is most useful when the database is open; it is also useful immediately after a crash, since it shows the backup status of the files at the time of the crash. You can use this information to determine whether you have left any tablespaces in backup.

Note: V\$BACKUP is not useful if the control file currently in use is a restored backup or a new control file created since the media failure occurred; a restored or re-created control file does not contain the information ORACLE needs to fill V\$BACKUP accurately. Also, if you have restored a backup of a file, that file's STATUS in V\$BACKUP reflects the backup status of the older version of the file, not the most current version's. Thus, this view might contain misleading information on restored files.

For example, the following query displays the current backup status of data files:

```
SELECT file#, status
FROM v$backup;
```

FILE#	STATUS
0011	INACTIVE
0012	INACTIVE
0013	ACTIVE
...	

In the STATUS column, “INACTIVE” indicates that the file is not currently being backed up and “ACTIVE” indicates that the file is marked as currently being backed up.

Privileges Required to Backup Online Tablespaces To back up online tablespaces, you must have the MANAGE TABLESPACE system privilege.

Backing Up Several Online Tablespaces If you have to back up several online tablespaces, use either of the following procedures:

- Back up the online tablespaces in parallel. For example, prepare all online tablespaces for backup:

```
ALTER TABLESPACE ts1 BEGIN BACKUP;
ALTER TABLESPACE ts2 BEGIN BACKUP;
ALTER TABLESPACE ts3 BEGIN BACKUP;
```

Next, back up all files of the online tablespaces and indicate that the online backups have been completed:

```
ALTER TABLESPACE ts1 END BACKUP;
ALTER TABLESPACE ts2 END BACKUP;
ALTER TABLESPACE ts3 END BACKUP;
```

- Backup the online tablespaces serially. For example, individually prepare, back up, and end the backup of each online tablespace:

```

ALTER TABLESPACE ts1 BEGIN BACKUP;
    backup files
ALTER TABLESPACE ts1 END BACKUP;
ALTER TABLESPACE ts2 BEGIN BACKUP;
    backup files
ALTER TABLESPACE ts2 END BACKUP;

```

The second option minimizes the time between ALTER TABLESPACE ... BEGIN/END BACKUP commands and is preferred:

- During online backups, more redo information is generated for the tablespace.
- If the instance crashes during online backup and the online log file that was written at BEGIN BACKUP time is reused, then archived redo log files are required for recovery.

Therefore, Oracle Corporation recommends that you back up online tablespaces serially.

Problems that Interrupt Online Tablespace Backups If a system or instance failure interrupts the backup of an online tablespace (after the ALTER TABLESPACE ... BEGIN BACKUP command is issued, but before the ALTER TABLESPACE ... END BACKUP command is issued), the database must be recovered using complete media recovery. See “Complete Media Recovery” on page 19-17 for more information.

Offline Tablespace and Offline Data File Backups

All or some of the data files of an individual tablespace can be backed up while the tablespace is offline. All other tablespaces of the database can remain open and available for system-wide use.

Note: You cannot take the SYSTEM tablespace or any tablespace with active rollback segments offline. Therefore, the following procedure cannot be used for such tablespaces.

To perform a backup of the offline data files of an offline tablespace, use the following steps:

1. Identify the data files of the offline tablespace.

Use the fully specified filename of the data file.

Before taking the tablespace offline, identify the names of its data files by querying the data dictionary view DBA_DATA_FILES. (See Step 1 of the previous section on page 18-9.)

2. Take the tablespace offline, using normal priority if possible.

If possible, use normal priority (not temporary or immediate) to guarantee that the tablespace can be subsequently brought online without the requirement for tablespace recovery.

To take a tablespace (and all associated data files) offline with normal priority, use the Set Tablespace Offline dialog box of SQL*DBA or the SQL command ALTER TABLESPACE with the OFFLINE parameter. For example, the following statement takes a tablespace named USERS offline normally:

```
ALTER TABLESPACE users OFFLINE NORMAL;
```

After a tablespace is taken offline with normal priority, all data files of the tablespace are closed. See page 7-9 for more information about online and offline tablespaces.

3. Back up the offline data files.

At this point, you can backup the data files of the offline tablespace:



- from within SQL*DBA, using the HOST command
- by exiting SQL*DBA and entering the operating system commands or starting the backup utility



See your installation or user's guide for more information about making operating system backups of files.

4. Bring the tablespace online. *(Optional)*

Bring the tablespace back online using either the Set Tablespace Online dialog box of SQL*DBA or the SQL command ALTER TABLESPACE with the ONLINE option. For example, the following statement brings an offline tablespace named USERS online:

```
ALTER TABLESPACE users ONLINE;
```

Note: If the tablespace was taken offline using temporary or immediate priority, the tablespace may not be able to be brought online unless tablespace recovery is performed. See Chapter 19 for more information about tablespace recovery.

After a tablespace is brought online, the data files of the tablespace are open and available for use.

Privileges Required to Backup Offline Tablespaces To take tablespaces offline and online, you must have the MANAGE TABLESPACE system privilege.

Control File Backups

Back up the control file of a database after making a structural modification to a database operating in ARCHIVELOG mode.

You can take a backup of a database's control file with the SQL command ALTER DATABASE with the BACKUP CONTROLFILE parameter. For example, the following statement backs up a database's control file:

```
ALTER DATABASE BACKUP CONTROLFILE TO 'filename' REUSE;
```

Here, filename is a fully specified filename that indicates the name of the new control file backup.

The REUSE option allows you to have the new control file overwrite a control file that currently exists. You cannot specify this option if you specify any of the following other options in the ALTER DATABASE BACKUP CONTROLFILE command:

- MAXLOGFILES
- MAXDATAFILES
- MAXINSTANCES
- MAXLOGMEMBERS
- MAXLOGHISTORY

Backing Up the Control File to the Trace File

The ALTER DATABASE BACKUP CONTROLFILE command has an option, TRACE, that can help you manage your control file and recover using it. This option causes ORACLE to write SQL commands to the database's trace file, rather than making a physical backup of the control file. These commands start up the database, re-create the control file, and recover and open the database appropriately, based on the current control file; each command is commented. Thus, you can copy the commands from the trace file into a script file, edit them as necessary, and use the script to recover the database if all copies of the control file are lost.

For example, assume the SALES database has three enabled threads, of which thread 2 is public and thread 3 is private; it also has multiplexed redo log files, and one offline and one online tablespace. The command could be used as follows:

```
ALTER DATABASE  
  BACKUP CONTROLFILE TO TRACE NORESETLOGS;
```

```
3-JUN-1992 17:54:47.27:  
# The following commands will create a new control file and use it  
# to open the database.  
# No data other than log history will be lost. Additional logs may  
# be required for media recovery of offline data files. Use this
```

```

# only if the current version of all online logs are available.
STARTUP NOMOUNT
CREATE CONTROLFILE REUSE DATABASE SALES NORESETLOGS ARCHIVELOG
    MAXLOGFILES 32
    MAXLOGMEMBERS 2
    MAXDATAFILES 32
    MAXINSTANCES 16
    MAXLOGHISTORY 1600
LOGFILE
    GROUP 1 'DISK$: [PROD.SALES.DB]LOG1T1.ORA' SIZE 100K,
    GROUP 2 (
        'DISK$: [PROD.SALES.DB]LOG2T1.ORA',
        'DISK$: [PROD.SALES.DB]G2.ORA'
    ) SIZE 100K,
    GROUP 3 'DISK$: [PROD.SALES.DB]LOG1T2.ORA' SIZE 100K,
    GROUP 4 'DISK$: [PROD.SALES.DB]LOG2T2.ORA' SIZE 100K,
    GROUP 5 (
        'DISK$: [PROD.SALES.DB]LOG1T3.ORA',
        'DISK$: [PROD.SALES.DB]G5.ORA'
    ) SIZE 100K
    GROUP 6 'DISK$: [PROD.SALES.DB]LOG2T3.ORA' SIZE 100K
DATAFILE
    'DISK$: [PROD.SALES.DB]DATABASE1.ORA' SIZE 2M,
    'DISK$: [PROD.SALES.DB]FILEA.ORA' SIZE 200K
;
# Take files offline to match current control file.
ALTER DATABASE DATAFILE 'DISK$: [PROD.SALES.DB]FILEA.ORA' OFFLINE;

# Recovery is required if any data files are restored backups,
# or if the last shutdown was not normal or immediate.
RECOVER DATABASE;

# All logs need archiving and a log switch is needed.
ALTER SYSTEM ARCHIVE LOG ALL;

# Database can now be opened normally.
ALTER DATABASE OPEN;

# Files in normal offline tablespaces are now named.
ALTER DATABASE RENAME FILE 'MISSING0002'
    TO 'DISK$: [PROD.SALES.DB]FILEB.ORA';

```

Using the command without NORESETLOGS produces the same output. Using the command with RESETLOGS produces a similar script that includes commands that recover the database and open the database, but that resets the redo logs on startup.

Recovering From a Failure During a Backup

If you crash during an online backup, you must recover the database when you next start it. (You do not need to issue an END BACKUP command once you have restarted the database.)

For this recovery, you do not need to restore previous backups of the data files. You might need archived redo logs.

You can determine whether files were in the process of being backed up by querying the table V\$BACKUP. See “Determining Data File Backup Status” on page 18-11.

Perform the recovery in either of two ways:

- Use the SQL*DBA Start Up Instance dialog with the Recover option, or the command STARTUP RECOVER to start and recover the database automatically.
- Start an instance, open and mount the database, and issue the statement RECOVER DATABASE,.

The first method is easier, because it causes ORACLE to perform recovery only if it is needed.

For information on starting the database, see Chapter 3. For information on recovering a database, see Chapter 19.

Using the Export and Import Utilities for Supplemental Database Protection

Export and Import are utilities that move ORACLE data in and out of ORACLE databases. Export writes data from an ORACLE database to an operating system file in a special format. Import reads Export files and restores the corresponding information into an existing database. Although Export and Import are designed for moving ORACLE data, you can also use them to supplement backups of data.



Utilities

Note: Both the Export and Import utilities are discussed in more detail in the *ORACLE7 Server Utilities User's Guide*.

Using Export

Export allows you to backup your database while it is open and available for use. It writes a read-consistent view of the database's objects to an operating system file.



Note: If you use Export for backup, all data must be exported in a logically consistent way so that the backup reflects a single point in time. No one should make changes to the database while the export takes place. Ideally, you should run the database in restricted mode while you export the data, so no regular users can access the data.

There are three Export modes:

User	exports all objects owned by a user
Table	exports all or specific tables owned by a user
Full Database	exports all objects of the database

This section discusses only the full database mode of operation.

There are also three types of exports:

- Incremental
- Cumulative
- Complete

Incremental Export

Only database data that has changed since the last incremental export is exported.

For example, if tables A, B, and C exist, and only table A's information has been modified since the last incremental export, only the changes to table A are exported.

Cumulative Exports

Only database data that has been changed since the last cumulative export is exported. Perform this type of export on a limited basis, such as once a week, to condense the information contained in numerous incremental exports.

For example, if tables A, B, and C exist, and only table A's and table B's information has been modified since the last cumulative export, only the changes to tables A and B are exported.

Complete Exports

All database data is exported. Perform this type of export on a limited basis, such as once a month, to export all data contained in a database.

Using Import

The Import utility allows you to restore the database information held in previously created Export files. It is the complement utility to Export.

To recover a database using Export files and the Import utility, the following operations are required:

1. Re-create the database structure, including the following items:

- all tablespaces
- all users

Note: These recreated structures should not have any objects in them.

2. Import the appropriate Export files to restore the database to the most current state possible. Depending on how your Export schedule is performed, imports of varying degrees will be necessary to restore a database.

For example, assume that the schedule illustrated in Figure 18-3 is used in exporting data from an ORACLE database.

FIGURE 18-3
A Typical Export Schedule

Day	1	2	3	4	5	6	7	8	9	10
Export Type	F	I	I	I	I	I	C	I	I	I

I = Incremental

C = Cumulative

F= Full (Complete)

A complete export was taken on Day 1, a cumulative export was taken every week, and incremental exports were taken daily.

To recover from a disk failure that occurs on Day 10, before the next incremental export is taken on Day 11, perform the following steps:

1. Recreate the database, including all tablespaces and users.
2. Import the complete database export taken on Day 1.
3. Import the cumulative database export taken on Day 7.
4. Import the incremental database exports taken on Days 8, 9, and 10.

RECOVERING A DATABASE

This chapter explains how to perform database media recovery.
Topics include:

- guidelines for media recovery
- recovering from media failure
 - restoring a full backup
 - complete media recovery
 - incomplete media recovery
- examples of media recovery

Guidelines and Prerequisites for Recovering from a Media Failure

The following sections provide some guidelines and prerequisites to consider before recovering a damaged database from a media failure.

Test Backup and Recovery Strategies

You should test your backup and recovery strategies in a test environment before you move to a production system, and continue to test your system regularly. By testing your backup and recovery strategies, you can test the thoroughness of your strategies and minimize problems before they occur in a real situation. Performing test recoveries on a regular basis ensures that your archiving and backup procedures work. It also helps you stay familiar with recovery procedures, so that you are less likely to make a mistake in a crisis.

Decide What Type of Recovery Operation Is Appropriate

Answer the following three questions after media failure damages a database:

1. What recovery operations are available?

The answer to this first question is determined by whether your database is archiving redo logs:

- If the database is operated in NOARCHIVELOG mode, usually the only option for recovery from the media failure is to restore the most recent full backup and re-enter all work performed since the backup. (If you have used Export to supplement regular backups, you can instead use Import to restore data.) Some special losses can be repaired more easily.
- If the database is operated in ARCHIVELOG mode, several recovery operations are available for reconstructing the damaged database.

2. What recovery operations are appropriate for this particular problem?

If the database is in ARCHIVELOG mode, several recovery operations are available to reconstruct a damaged database to a transaction consistent state prior to the time of the media failure or user error. See “Examples of Media Failures and Appropriate Recovery Procedures” on page 19-27 for a detailed list of the different types of problems that media failures can cause and the appropriate methods of recovery from each type of problem.

3. Is the damaged database part of a distributed database?

If so, database recovery may need to be coordinated among the nodes of the distributed database. See “Coordinate Distributed Recovery” below for more information.

Correct or Avoid a Disk Problem

The goal of database recovery is to reopen a database for normal operation as soon as possible. If a media failure has occurred because of a hardware problem, the problem should be repaired as soon as possible; however, database recovery is not dependent on the resolution of a long-lasting hardware problem that caused a media failure. Files stored on the damaged device can be restored to other storage devices and the file relocation can be accounted for using the procedures in the following sections:

Type of File	Section Name	Page #
Data File	Renaming and Relocating Data Files for Tablespaces	7-13
Online Redo Log File	Renaming and Relocating Online Redo Log Members	5-7
Control File	Creating Additional Copies of the Control File and Renaming or Relocating Control Files	6-4

Coordinate Distributed Recovery

The ORACLE distributed database architecture is autonomous in nature. Therefore, depending on the type of recovery operation selected for any single damaged database, recovery operations may or may not have to be coordinated globally among all databases in the distributed database system. Table 19-1 summarizes the different types of recovery operations and whether or not coordination among nodes of a distributed database system is required.

TABLE 19-1
Database Recovery in a
Distributed Database System

Type of Recovery Operation	Implication for Distributed Database System
Restoring a full backup for a database that was never accessed (updated or queried) from a remote node	Use non-coordinated, autonomous database recovery.
Restoring a full backup for a database that was accessed by a remote node	Shut down all databases and restore them using the same, coordinated full backup.
Complete media recovery of one or more databases in a distributed database	Use non-coordinated, autonomous database recovery.
Incomplete media recovery of a database that was never accessed by a remote node	Use non-coordinated, autonomous database recovery.
Incomplete media recovery of a database that was accessed from a remote node	Use coordinated incomplete media recovery to the same global point-in-time for all databases in the distributed database.

Coordinate Time-Based and Change-Based Distributed Database Recovery

In special circumstances, one node in a distributed database may require recovery to a point-in-time in the past. To preserve global data consistency, it is often necessary that all other nodes in the system also be recovered to the same point-in-time. This is called “coordinated time-based distributed database recovery.”

The procedures for autonomous time-based and change-based recovery are given in the following sections of this chapter. However, coordinated time-based distributed database recovery requires some extra steps to correct for the variance in time usually observed among different computers. Therefore, in conjunction with the normal procedures of time-based and change-based recovery, use the following steps to coordinate distributed time-based recovery among many nodes in a distributed database system:

1. Recover the database that is requiring the recovery operation using time-based recovery. For example, if a database needs to be recovered because of a user error (such as an accidental table drop), recover this database first using time-based recovery. Do not recover the other databases yet.

2. After you have recovered the database and opened it using RESETLOGS, look in the ALERT file of the database for the RESETLOGS message.

If the message says, “RESETLOGS after complete recovery through change nnnnnnnn,” you have applied all the changes in the database and actually performed a complete recovery. Do not recover any of the other databases in the distributed system, or you will unnecessarily remove changes in them. Recovery is finished.

If the reset message says, “RESETLOGS after incomplete recovery UNTIL CHANGE nnnnnnnn,” you have successfully performed an incomplete recovery. Record the change number from the message and proceed to the next step.

3. Recover all other databases in the distributed database system using change-based recovery, specifying the change number (SCN) from Step 2.

Recover Databases with Snapshots

If a master database is independently recovered to a past point-in-time (that is, coordinated time-based distributed database recovery is not performed), any dependent remote snapshot that refreshed in the interval of lost time will be inconsistent with its master table. In this case, the administrator of the master database should instruct the remote administrators to perform a complete refresh of any inconsistent snapshot.

Performing Media Recovery

The following sections explain the steps necessary to recover from media failures by restoring a full backup, or by using complete or incomplete media recovery operations.

The following sections explain media recovery operations without regard to the media failure itself or the types of files that were damaged by the media failure. Therefore, the following sections should not be used alone to understand the procedures necessary to recover from a media failure; “Examples of Media Failures and Appropriate Recovery Procedures” on page 19-27 provides the appropriate methods of recovery from each type of problem.

Issues Common to All Media Recovery Operations

The following sections discuss topics common to all complete and incomplete media recovery operations (but not restoring a full backup). These topics should be understood before proceeding to the next sections.

Using Messages that Indicate Media Failures

Error messages like those that follow indicate the need for media recovery:

ORA-00204	“error in reading control file ‘ <i>name</i> ’ (block <i>num</i> , # blocks <i>num</i>)”
ORA-01113	“file <i>name</i> needs recovery”
ORA-01168	“file <i>name</i> : bad physical block size of <i>num</i> bytes expecting <i>num</i> ”
ORA-01178	“file <i>name</i> created before last CREATE CONTROLFILE, cannot recreate”



Message

See the *ORACLE7ServerMessages and Codes Manual* for more information about these and other messages returned by ORACLE.

Determining Whether a File Needs Recovery

To determine which files need recovery, you can often use the table V\$RECOVER_FILE. This table lists all files that need recovery, along with information on why they need recovery.

Note: V\$RECOVER_FILE is most useful when the database is closed, because when the database is open the table contains information about offline files only. The table is not useful if the control file currently in use is a restored backup or a new control file created since the media failure occurred; a restored or re-created control file does not contain the information ORACLE needs to fill V\$RECOVER_FILE accurately.

For example, the following query displays the file ID numbers of data files that require recovery:

```
SELECT file#, online, error
FROM v$recover_file;
```

FILE#	ONLINE	ERROR
0014	ONLINE	
0018	ONLINE	FILE NOT FOUND
0032	OFFLINE	OFFLINE NORMAL
...		

To find the name of a file based on its file number, use the data dictionary view V\$DATA_FILE, which contains the file’s NAME as well as FILE#.

Deciding Which Recovery Action to Perform

Table 19-2 indicates what to do to recover from the loss of files due to a single media failure. The procedures indicated are detailed in the following sections.

TABLE 19-2
Recovering from
the Loss of Files

Types of Damaged Files				Archiving Mode	
Data File	Online Redo Log File	Archived Redo Log File	Control File	ARCHIVELOG Mode	NOARCHIVELOG Mode
✓				Use complete media recovery (page 19-17).	Restore a full backup (page 19-16).
	✓			Recover the missing files (page 19-29).	Recover the missing files (page 19-29).
		✓		Take a new backup of all data files (page 19-34).	(does not apply)
			✓	Recover the missing files (page 19-34).	Recover the missing files (page 19-34).
✓	✓	✓	✓	Recover the control files, also recovering other files (page 19-34).	(does not apply)
✓	✓	✓		Use incomplete media recovery (page 19-29).	(does not apply)
✓		✓	✓	Recover the control files, also recovering data files (page 19-34).	(does not apply)
✓	✓		✓	Recover the control files, also recovering other files (page 19-34).	Restore a full backup (page 19-16).
✓	✓			Use incomplete media recovery (page 19-29).	Restore a full backup (page 19-16).
✓		✓		Use incomplete media recovery (page 19-34).	(does not apply)
✓			✓	Recover the control files, also recovering data files (page 19-34).	Restore a full backup (page 19-16).
	✓	✓	✓	Recover the control files, resetting the redo log (page 19-34).	(does not apply)
	✓	✓		Use incomplete media recovery (page 19-29).	(does not apply)
	✓		✓	Recover the control files, resetting the redo log (page 19-34).	If the database was shut down normally, re-create the control file (page 19-34). Otherwise, restore a full backup (page 19-16).
		✓	✓	Recover the control files, using incomplete media recovery (page 19-34).	(does not apply)

Restoring Damaged Data Files

If a media failure permanently damages one or more data files of a database, you must restore backups of the damaged data files before you can recover the damaged files.

Relocating Damaged Files If a damaged data file cannot be restored to its original location (for example, a disk must be replaced, so the files are restored to an alternate disk), the new locations of these files must be indicated to the control file of the associated database. Therefore, use the procedure given in “Renaming and Relocating Data Files” on page 7-13, as necessary. (This procedure is referenced throughout the rest of this chapter.)

Recovering a Data File Without a Backup If a data file is damaged and no backup of the file is available, the data file can still be recovered, provided that all log files written since the creation of the original data file are available, and the control file contains the name of the damaged file (that is, the control file is current, or is a backup taken after the damaged data file was added to the database).

Use the CREATE DATAFILE clause of the ALTER DATABASE command to create a new, empty data file, replacing a damaged data file that has no corresponding backup. For example, assume that the data file “disk1:users1” has been damaged and no backup is available. The following statement recreates the original data file (same size) on disk 2:

```
ALTER DATABASE CREATE DATAFILE 'disk2:users1' AS 'disk1:users1';
```

Note: The old data file is implicitly renamed as the new data file when an ALTER DATABASE CREATE DATAFILE statement is issued.

This statement causes ORACLE to create an empty file that matches the lost file. (ORACLE looks at information in the control file and the data dictionary to obtain size information.) You must next perform media recovery on the empty data file. All archived redo logs written since the original data file was created must be mounted and reapplied to the new, empty version of the lost data file during recovery. If the database was created in NOARCHIVELOG mode, the original data files of the SYSTEM tablespace cannot be restored using an ALTER DATABASE CREATE DATAFILE statement because the necessary archived redo logs are not available.

Restoring Necessary Archived Redo Log Files

All archived redo log files required for the pending media recovery eventually need to be on disk so that they are readily available to ORACLE.

To determine which archived redo log files you need, you can often use the tables V\$LOG_HISTORY and V\$RECOVERY_LOG.

V\$LOG_HISTORY lists all of the archived logs, including the names they are most likely to have, given the current archived log file naming scheme (as set by the parameter LOG_ARCHIVE_FORMAT). V\$RECOVERY_LOG lists only the archived redo logs that ORACLE believes it needs to perform recovery; it also includes the most likely names of the files, using LOG_ARCHIVE_FORMAT. (See Appendix B for more information on these tables.)

If space is available, restore all of the required archived redo log files to the location currently specified by the initialization parameter LOG_ARCHIVE_DEST. By doing this, you allow ORACLE to locate automatically the correct archived redo log file when required during media recovery. If sufficient space is not available at the location indicated by LOG_ARCHIVE_DEST, you can restore some or all of the required archived redo log files to any disk accessible to ORACLE. In this case, you can specify the location of the archived redo log files before or during media recovery.

After an archived log is applied, you can delete the restored copy of the archived redo log file to free disk space. However, make sure that a copy of each archived log group still exists on offline storage.

Starting Media Recovery

If a damaged database is operating in ARCHIVELOG mode, it is a candidate for either complete media recovery or incomplete media recovery operations. To begin media recovery operations, use one of the following options of SQL*DBA:

- the Recover Closed Database dialog box
- the Recover Offline Tablespaces dialog box
- the Recover Data File dialog box
- the SQL*DBA RECOVER command
- the SQL command ALTER DATABASE
- the Start Up Instance dialog box or the SQL*DBA STARTUP command, with the RECOVER option



Utilities

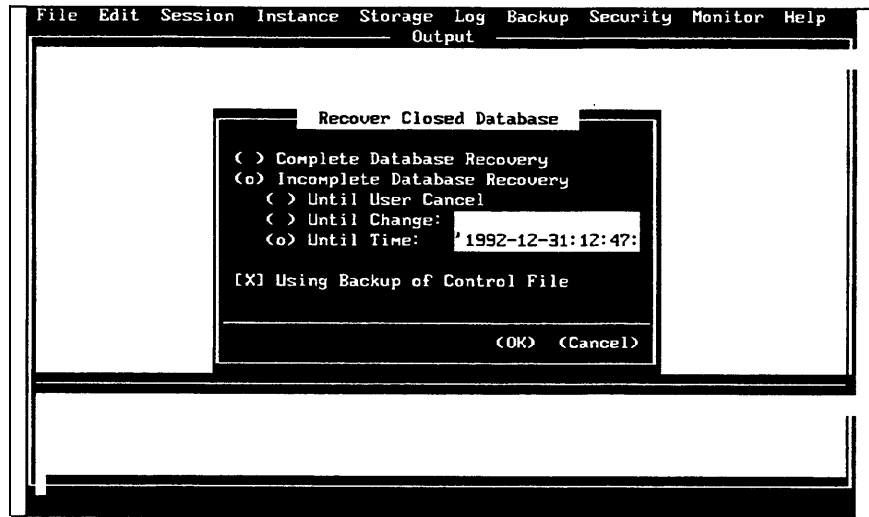


SQL

This section provides examples for each of the above options. (For information about SQL*DBA, see the *ORACLE7 Server Utilities User's Guide*, and for complete information about the ALTER DATABASE command, see the *ORACLE7 Server SQL Language Reference Manual*.)

Recovering a Closed Database After the database is mounted but closed, start closed database recovery (complete or incomplete) using either SQL*DBA'S Recover Closed Database dialog box or the RECOVER command with the DATABASE parameter. Figure 19-1 shows the Recover Closed Database dialog box that begins time-based recovery using a backup version of the control file.

FIGURE 19-1
The Recover Closed
Database Dialog

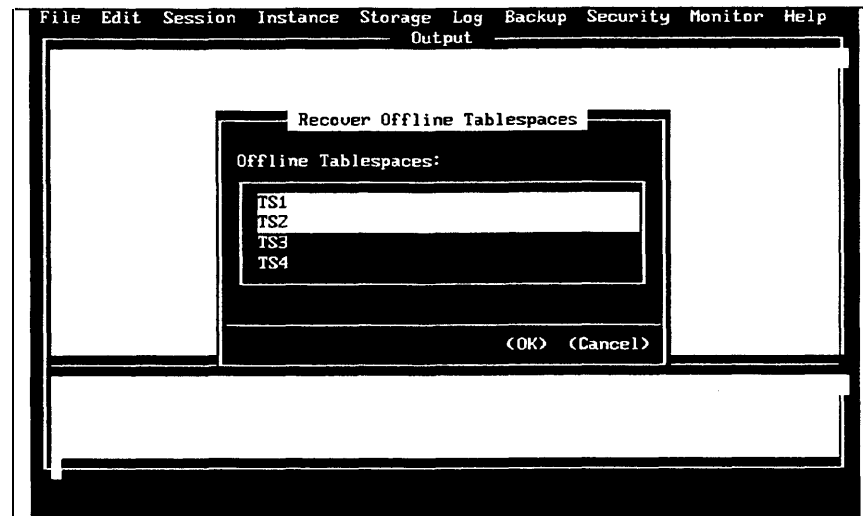


The following statement is the command equivalent of the Recover Closed Database dialog box in Figure 19-1:

```
RECOVER DATABASE
    UNTIL '1992-12-31:12:47:30' USING BACKUP CONTROLFILE;
```

Recovering an Offline Tablespace in an Open Database After the tablespaces of interest are taken offline (see “Taking Tablespaces Offline” on page 7-10), you can start open database-offline tablespace recovery using either of the following options of SQL*DBA: the Recover Offline Tablespaces dialog box or the RECOVER command with the TABLESPACE parameter. Either option begins complete media recovery of one or more offline tablespaces; the remainder of the database may be open and online for normal database operation. Figure 19-2 shows the Recover Offline Tablespaces dialog box.

FIGURE 19-2
The Recover Offline
Tablespaces Dialog

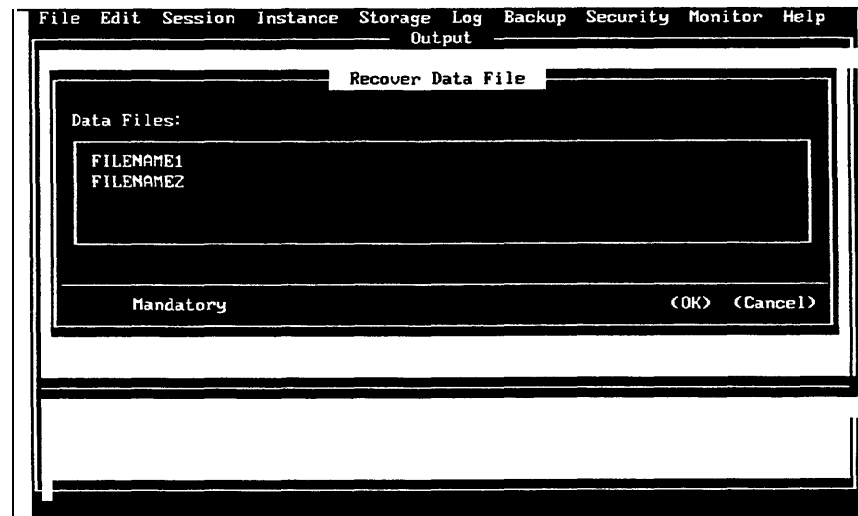


The following statement is the command equivalent of the Recover Offline Tablespaces dialog box in Figure 19-2:

```
RECOVER TABLESPACE ts1, ts2;
```

After the tablespaces that contain the damaged files have been taken offline and you are positive the associated data files are also offline (check in V\$DATAFILE to see the file's status), recover selected data files using the Recover Data File dialog box or the RECOVER command with the DATAFILE parameter. Figure 19-3 shows the Recover Data File dialog box.

FIGURE 19-3
The Recover Data File Dialog



The following statement is the command equivalent of the Recover Data File dialog box in Figure 19-3:

```
RECOVER DATAFILE 'filename1', 'filename2';
```

The SQL command equivalent of either of the SQL*DBA media recovery options is the SQL command ALTER DATABASE command with the RECOVER clause. In most cases, database recovery should be performed using SQL*DBA; its interface prompts you for information and returns messages from the system. However, if you want to design your own recovery application using SQL commands, the ALTER DATABASE command is available.

Starting Recovery During Instance Startup Complete media recovery can be started using the Start Up Instance dialog box with the Recover check box or the STARTUP command with the RECOVER option when using SQL*DBA. Once an instance is started and the database is mounted, complete media recovery proceeds as described in “Complete Media Recovery” on page 19-17. See Chapter 3 and the *ORACLE7 Server Utilities User's Guide* for more information about the Start Up Instance dialog box and STARTUP command.



Utilities

Privileges Required to Start Media Recovery To start any type of media recovery, you must be connected as INTERNAL. All recovery sessions must be compatible; one session cannot start complete media recovery while another performs incomplete media recovery. Also, you cannot start media recovery if you are connected to the database via a multi-threaded server process. See Chapter 4 for more information about multi-threaded server processes.

Applying Redo Log Files

During complete or incomplete media recovery, redo log files (online and archived) are applied to the data files during the roll forward phase of media recovery. As a log file is needed, ORACLE suggests the name of the file. For example, if you are using SQL*DBA, it returns the following lines and prompt:

```
ORA-00279: Change ##### generated at DD/MM/YY HH:MM:SS needed for thread #
ORA-00289: Suggestion : logfile
ORA-00280: Change ##### for thread # is in sequence #
Specify log: [<RET> for suggested | AUTO | FROM logsource | CANCEL]
```

Note: Similar messages are returned when using an ALTER DATABASE ... RECOVER statement; however, no prompt is returned.

The following sections describe how log files can be applied in different environments.

Suggested Log Filenames ORACLE suggests log filenames by concatenating the current values of the initialization parameters

LOG_ARCHIVE_DEST and LOG_ARCHIVE_FORMAT and using some information from the control file. Therefore, if all the required archived log files are mounted at LOG_ARCHIVE_DEST and the value for LOG_ARCHIVE_FORMAT is never altered, ORACLE can suggest and apply log files to complete media recovery automatically without intervention from the administrator. If the location specified by LOG_ARCHIVE_DEST is not available (for example, because of media failure), you can change the value for this parameter, mount the log files to the new location, and start a new instance before beginning media recovery.

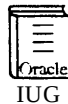
In some cases, you might want to override the current setting for LOG_ARCHIVE_DEST as a source for log files. For example, assume that a database is open and an offline tablespace must be recovered, but not enough space is available to mount the necessary log files at the location specified by LOG_ARCHIVE_DEST. In this case, the necessary log files can be mounted to an alternate location and the alternate location is indicated to ORACLE for the recovery operation. To specify the location where required log files can be found, use the LOGSOURCE parameter of the SQL*DBA command SET when using SQL*DBA or use the RECOVER ... FROM parameter of the SQL command ALTER DATABASE when using SQL commands.

Note: Overriding the log source does not affect the archive log destination for filled online groups being archived.

Consider overriding the current setting for LOG_ARCHIVE_DEST when not enough space is available to mount all the required log files at any one location. In this case, you can set the log file source to an operating system variable (such as a logical or an environment variable) that acts as a search path to several locations; however, such functionality is operating system dependent; see your installation or user's guide.



Applying Log Files when Using SQL*DBA If the suggested archived redo log file is correct, apply the suggested archived redo log by pressing the Enter or Return key; you do not have to specify a filename unless the suggested file is incorrect. See your installation or user's guide for an example of this procedure. Once a filename is provided, ORACLE then applies the redo log file to roll forward the restored data files.



When using SQL*DBA, you can have ORACLE automatically apply the redo log files that it suggests by using either of two options:

- Before starting media recovery, issue the following SQL*DBA statement to turn automatic recovery on:

SET AUTORECOVERY ON;

Automatic application of the suggested redo log starts once recovery begins.
- After media recovery is started, enter “auto” when prompted for a redo log file. Automatic application of the suggested redo log starts from this point forward.

Suggested redo log files are automatically applied until a suggested redo log is incorrect or recovery is complete. Online redo log files may need to be manually specified only when cancel-based recovery or a backup of the control file is being used.

Application of Log Files When Using SQL Commands Application of redo log files is similar to the behavior described in the previous section. However, a prompt for log files is not returned once media recovery is started. Instead, you must provide the correct log file using an ALTER DATABASE RECOVER LOGFILE statement. For example, if a message suggests LOG1.ARC, you can apply the suggestion using the following statement:

```
ALTER DATABASE RECOVER LOGFILE 'log1.arc';
```

As a result, restoring a tablespace requires several statements, as indicated in the following example: (DBA input is highlighted in bold; variable information is shown in italics.)

```
> ALTER DATABASE RECOVER TABLESPACE users;
ORA-00279: Change ##### generated at DD/MM/YY HH:MM:SS needed for thread #
ORA-00289: Suggestion : logfile1
ORA-00280: Change ##### for thread # is in sequence #
> ALTER DATABASE RECOVER LOGFILE 'logfile1';
ORA-00279 : Change ##### generated at DD/MM/YY HH:MM:SS needed for thread #
ORA-00289: Suggestion : logfile2
ORA-00280: Change ##### for thread # is in sequence #
> ALTER DATABASE RECOVER LOGFILE 'logfile2';
(Repeat until all logs are applied.)
Statement processed.
> ALTER TABLESPACE users ONLINE;
Statement processed.
```

Note: This example assumes that the backup files have been restored, and that the user is connected as INTERNAL.

Similarly to the method you would use with SQL*DBA, automatic application of the redo logs can be started with the following statements, before and during recovery, respectively:

```
ALTER DATABASE RECOVER AUTOMATIC ...;
```

```
ALTER DATABASE RECOVER AUTOMATIC LOGFILE suggested_log_file;
```

The following is an example of using the first statement:

```
> ALTER DATABASE RECOVER AUTOMATIC TABLESPACE users;
Statement processed.
> ALTER TABLESPACE users ONLINE;
Statement processed.
```

Note: This example assumes that the backup files have been restored, and that the user is connected as INTERNAL.

Similarly, the following is an example of using the “ALTER DATABASE RECOVER AUTOMATIC LOGFILE” statement:

```
> ALTER DATABASE RECOVER TABLESPACE users;
ORA-00279: Change ##### generated at DD/MM/YY HH:MM:SS needed for thread #
ORA-00289: Suggestion : logfile1
ORA-00280: Change ##### for thread # is in sequence #
> ALTER DATABASE RECOVER AUTOMATIC LOGFILE 'logfile1';
Statement processed.
> ALTER TABLESPACE users ONLINE;
Statement processed.
```

Note: This example assumes that the backup files have been restored, and that the user is connected as INTERNAL.

Successful Application of Redo Logs If you are using SQL*DBA's recovery options (not SQL statements), each time ORACLE finishes applying a redo log file, the following message is returned:

Log applied.

Make sure that the message "Log applied" is returned after each application of a redo logfile. If the suggested file is incorrect or you provide an incorrect filename, an error message is returned instead. If you see an error message instead of "Log applied," a redo log file needed for recovery has not been applied. Recovery cannot continue until the needed redo log file is applied.

If an error message is returned after supplying a redo log filename, the kind of error message that is returned depends upon the kind of error that has been detected:

- If the error message says that the file cannot be found, you may have typed in the wrong filename. Re-enter the correct filename.
- If the redo log file is found, but cannot be opened, then it may be locked. After unlocking the redo log file, re-enter the filename.
- If a redo log file is found and opened, but cannot be read, an I/O error is returned. In this case, the redo log file may have been only partially written or may contain corrupt data. If you can locate an uncorrupted or complete copy of the log, you can simply apply that copy; you do not need to restart recovery. Otherwise, if no other copy of the log exists and if you know the time of the last valid redo entry, you can perform time-based or change-based recovery; in this case, you must restart recovery from the beginning, including restoring backups.

Interrupting Media Recovery

If you start a media recovery operation and then must interrupt it for some reason (for example, because a recovery operation must end for the night and resume the next morning), you can interrupt recovery at any time by using either of these procedures:

- Enter the word CANCEL when prompted for a redo log file.
- If you must abort the procedure of recovering an individual data file or automated recovery is in progress, use your operating system's interrupt signal.

After recovery is canceled, it must be completed before opening a database for normal operation. To resume recovery, restart it; recovery resumes where it left off if it was canceled. If you want to restart recovery beginning with an earlier redo log file or a different backup, restart the entire recovery process by restoring the necessary backups again.

Restoring a Full Backup

If a database is operating in NOARCHIVELOG mode and a media failure damages part or all of the database, usually the only option for recovering the database is to restore the most recent full backup. (If you are using Export to supplement regular backups, you can instead restore the database by importing an exported backup of the database; see "Using the Export and Import Utilities for Supplemental Database Protection" on page 18-17.)

Note: If a database operating in ARCHIVELOG mode is damaged by a media failure, it can be recovered by restoring the data files from the most recent full backup; however, because the database's online redo

log is archived, complete or incomplete recovery from the media failure is normally used to reconstruct the database and minimize the amount of work lost.

To recover a damaged database by restoring the most recent full backup, use the following steps:

1. If the database is open, shut it down using either of the following options of SQL*DBA: the Abort Instance option of the Shut Down menu or the SHUTDOWN command with the ABORT option.
2. Repair the hardware problems that caused the database failure. If the hardware problem can be repaired quickly, follow Step 3a; if a hardware failure cannot be repaired quickly and the database must be reopened as soon as possible, follow Steps 3b through 3e.
3. **If the hardware problem has been repaired, follow Step 3a. If the hardware problem has not been corrected and alternative disks are used to store the restored database, follow Steps 3b through 3e:**
 - a. Restore the most recent full backup, including **all** data files, online redo log files, and control files. Do not restore just the files damaged by the media failure; **all** files must be restored from the most recent full backup to guarantee that the entire database corresponds to a single point in time. Proceed to Step 4.
 - b. Restore all data files, online redo log files, control files, and the parameter file of the database to operable disks, as desired.
 - c. Edit the restored parameter file to indicate the new location of the control files, if applicable.
 - d. Start an instance using the restored and edited parameter file and mount, but do not open, the database.
 - e. Immediately perform the steps necessary to record the relocation of the restored data files and online redo log files (from Step 3b), as described in “Renaming and Relocating Data Files” and “Renaming and Relocating Online Redo Log Members” on pages 7-13 and 5-7, respectively.
4. Reopen the database.

Complete Media Recovery

The following section explains the steps necessary to complete the different types of complete media recovery operations. The following sections should not be used alone to understand the procedures necessary to recover from a media failure; “Examples of Media Failures and Appropriate Recovery Procedures” on page 19-27 furnishes a detailed list of the different types of problems that media failures can

cause and provides the appropriate methods of recovery from each type of problem.

The following sections describe the following types of complete media recovery procedures:

- closed database recovery
- open database, offline tablespace recovery

Performing Closed Database Recovery

Use the following steps to perform closed database recovery, either of all damaged data files in one operation or individual recovery of each damaged data file in separate operations:

1. If the database is open, shut it down using either the Abort Instance option of the Shut Down menu or SQL*DBA, or the SHUTDOWN command with the ABORT option.
2. Correct the hardware problem that caused the media failure. If the hardware problem cannot be repaired quickly, database recovery can proceed by restoring damaged files to an alternative storage device.
3. If files are permanently damaged, restore the most recent backup files (taken as part of either a full backup or partial backup) of only the data files damaged by the media failure—do not restore any data files not damaged by the media failure or any online redo log files. If the hardware problem has been repaired and damaged data files can be restored to their original locations, do so, and skip Step 6 of this procedure; if the hardware problem persists, restore the data files to an alternative storage device of the database server and perform Step 6 when you come to it.

Note: If you do not have a backup of a specific data file, you might be able to create a replacement file, empty of contents, that can be recovered. See “Restoring Damaged Data Files” on page 19-8 for more information about creating data files.

4. Start SQL*DBA and connect to ORACLE as INTERNAL.
5. Start a new instance and mount, but do not open, the database. This operation can be performed using either of the following options of SQL*DBA the Start Up Instance dialog box with the Mount radio button selected or the STARTUP command with the MOUNT option.

6. If one or more damaged data files were restored to alternative locations in Step 3, the relocation of these files must be indicated to the control file of the associated database. Therefore, use the operation discussed in “Renaming and Relocating Data Files” on page 7-13, as necessary.

7. All data files that are to be recovered must be online during a complete media recovery. You can determine the names of all data files using the list of data files that accompanies the current control file (if available, see “Listing Database Files Before Backup” on page 18-6 for more information about such lists) or query the V\$DATAFILE view. Then, use the ALTER DATABASE command and the DATAFILE ONLINE option to make sure that all data files of the database are online. For example, to guarantee that a data file named USERS1 (a fully specified filename) is online, enter the following statement:

```
ALTER DATABASE DATAFILE 'users1' ONLINE;
```

If a data file is specified that is already online, ORACLE ignores the statement.

8. Depending on the type of closed database recovery desired, follow either Step 8a or 8b.

a. Use either of the following options of SQL*DBA to start closed database recovery of all damaged data files in one step: the Recover Closed Database dialog box or an equivalent RECOVER DATABASE statement. Proceed to Step 9 to continue the closed database recovery operation.

b. Use either of the following options of SQL*DBA to start closed database recovery of an individual damaged data file: the Recover Offline Data Files dialog box or an equivalent RECOVER DATAFILE statement.

Note: Multiple, concurrent sessions can be used to recover a few damaged data files simultaneously with each session. If many data files need recovery, this method can potentially expedite the media recovery process. If this option is used, make sure that all damaged data files are recovered. Each session behaves as in Steps 9 and 10.

9. ORACLE now begins the roll forward phase of media recovery by applying the necessary redo log files (archived and online) to reconstruct the restored data files. If automated, the suggested log files are automatically applied; if not automated, ORACLE prompts for the required redo log file. See “Applying Redo Log Files” on page 19-12 for more information.

ORACLE continues to apply archived redo log files until all required archived redo log files have been applied to the restored data files. The online redo log files are then automatically applied to the restored data files to complete media recovery, as indicated by the following message:

Media recovery complete.

Note: If no archived redo log files are required for complete media recovery, ORACLE does not prompt for any. Instead, all necessary online redo log files are applied, and the “Media recovery complete” message is returned.

If you have completed the above steps, the closed database is now recovered up to the moment that media failure occurred. You can now open the database using the SQL command ALTER DATABASE with the OPEN option.

Performing Open Database-Offline Tablespace Recovery

Use the following steps to perform either open database-offline tablespace recovery or open database-offline tablespace-individual data file recovery. At this point, an open database has experienced a media failure, and the database remains open while the undamaged data files remain online and available for use and the damaged data files are automatically taken offline by ORACLE.

Note: Neither of the following procedures can be used to perform complete media recovery on the data files of the SYSTEM tablespace. If the media failure damages any data files of the SYSTEM tablespace, ORACLE automatically shuts down the database. To proceed with complete media recovery, follow the procedure in the previous section, “Performing Closed Database Recovery.”

1. The starting point for this recovery operation can vary, depending on whether you have left the database open after the media failure has occurred.
 - a. **If the database has been shut down,** start a new instance, and mount and open the database. This operation can be performed using either of the following options of SQL*DBA: the Start Up Instance dialog box with the Open radio button selected or the STARTUP command with the OPEN option. Once the database is open, take all tablespaces offline that contain damaged data files. Follow Step 1b for this procedure.

- b. **If the database is open** with only damaged data files of the database offline, take all tablespaces offline that contain damaged data files. ORACLE should be indicating the damaged data files by the error messages it returns. Tablespaces can be taken offline using either the Set Tablespace Offline dialog box of SQL*DBA or the SQL command ALTER TABLESPACE with the OFFLINE option, as described in “Taking Tablespaces Offline” on page 7-10. If possible, take the damaged tablespaces offline with temporary priority (to minimize the recovery that is required). Also, pay attention to the special instructions in this referenced section concerning tablespaces that contain active rollback segments. After this step is completed, proceed to Step 2.
2. Correct the hardware problem that caused the media failure. If the hardware problem cannot be repaired quickly, database recovery can proceed by restoring damaged files to an alternative storage device.
3. If files are permanently damaged, restore the most recent backup files (taken as part of either a full backup or partial backup) of **only** the data files damaged by the media failure—do not restore any data files not damaged by the media failure, or any online redo log files or control files. If the hardware problem has been repaired and the data files can be restored to their original locations, do so, and skip Step 4 of this procedure; if the hardware problem persists, restore the data files to an alternative storage device of the database server and perform Step 4 next.

Note: If you do not have a backup of a specific data file, you can create a replacement file, empty of contents, that can be recovered. See “Restoring Damaged Data Files” on page 19-8 for more information about creating data files.
4. If one or more damaged data files were restored to alternative locations in Step 3, the relocation of these files must be indicated to the control file of the associated database. Therefore, use the operation discussed in “Renaming and Relocating Data Files” on page 7-13, as necessary.
5. **Depending on the type of complete media recovery procedure desired, follow Step 5a or 5b after connecting as INTERNAL.**
 - a. Use either of the following options of SQL*DBA to start offline tablespace recovery of all damaged data files in one or more offline tablespaces using one step: the Recover Offline Tablespaces dialog box or an equivalent RECOVER TABLESPACE statement. Skip to Step 6 to continue the offline tablespace recovery operation.

- b. Use either of the following options of SQL*DBA to start recovery of an individual damaged data file in an offline tablespace: the Recover Offline Data Files dialog box or an equivalent RECOVER DATAFILE statement.

Note: Multiple, concurrent sessions can be used to recover a few damaged data files simultaneously in each session, if the data files are on separate disks. If many data files need recovery, this method can potentially expedite the media recovery process. If this option is used, make sure that all damaged data files are recovered. Each session behaves as in Steps 6 and 7.

6. ORACLE now begins the roll forward phase of media recovery by applying the necessary redo log files (archived and online) to reconstruct the restored data files. If automated, the suggested log files are automatically applied; if not automated, ORACLE prompts for the required redo log file. See “Applying Redo Log Files” on page 19-12 for more information.

ORACLE continues to apply archived redo log files until all required archived redo log files have been applied to the restored data files. The online redo log files are then automatically applied to the restored data files to complete media recovery, as indicated by the following message:

Media recovery complete.

Note: If no archived redo log files are required for complete media recovery, ORACLE does not prompt for any. Instead, all necessary online redo log files are applied, and the “Media recovery complete” message is returned.

7. The damaged tablespaces of the open database are now recovered up to the moment that media failure occurred. The offline tablespaces can be brought online using either the Set Tablespace Online dialog box of SQL*DBA or the SQL command ALTER TABLESPACE with the ONLINE option.

Incomplete Media Recovery

The following section explains the steps necessary to complete the different types of incomplete media recovery operations: time-based, change-based, and cancel-based recovery. The following section should not be used alone to understand the procedures necessary to recover from a media failure; “Examples of Media Failures and Appropriate Recovery Procedures” on page 19-27 furnishes a detailed list of the different types of problems that media failures can cause and provides the appropriate methods of recovery from each type of problem.

Performing Cancel-Based,
Time-Based, or
Change-Based Recovery

Use the following steps to perform cancel-based, time-based, or change-based recovery:

1. If the database is still open and it has been decided that incomplete media recovery is necessary, shut down the database using either of the following options of SQL*DBA: the Abort Instance option of the Shut Down menu or the SHUTDOWN command with the ABORT option.
2. Make a full backup of the database (all data files, online redo log files, a control file, and the parameter files of the database) as a precautionary measure, in case an error is made during the recovery procedure. (This protects against any problems that might arise in restoring backups in step 5.)
3. If a media failure has occurred, correct the hardware problem that caused the media failure.
4. If the current control files do not match the physical structure (the set of files) of the database at the intended time of recovery (for example, if a data file has been added subsequent to the time to which you intend to recover), then you should restore a backup of the control file that reflects the database's physical file structure (contains only and all the names of data files and online redo log files) at the intended time to which incomplete media recovery is to finish. Review the list of files that corresponds to the current control file and each control file backup to determine the correct control file to use and, if necessary, replace all current control files of the database with the correct control file backup. (You can, alternatively, create a new control file to replace the missing one; see "Creating Additional Control Files and Renaming or Relocating Control Files" on page 6-4.)

Note: If any control file of the database cannot function or be replaced with a control file backup because a hardware problem that caused a media failure persists, you must edit the parameter file associated with the database to modify the CONTROL_FILES parameter. See the procedure on page 6-4.

5. Restore backup files (taken as part of either a full backup or partial backup) of **all** the data files of the database. All backup files used to replace existing data files must have been taken prior to the intended time of recovery. For example, if you intend to recover to redo log sequence number 38, then restore all data files with backups completed prior to redo log sequence number 38.

Note: If you do not have a backup of a specific data file, you can create a replacement file, empty of contents, that can be recovered.

See “Restoring Damaged Data Files” on page 19-8 for more information about creating data files.

Note: If a data file was added after the intended time of recovery, it is not necessary to restore a backup for this file, as it will no longer be used for the database after recovery is complete.

If the hardware problem that caused a media failure has been repaired and all data files can be restored to their original locations, do so, and skip Step 8 of this procedure; if a hardware problem persists, restore damaged data files to an alternative storage device and perform Step 8 when you come to it.

6. Start SQL*DBA and connect to ORACLE as INTERNAL.
7. Start anew instance and mount the database. This operation can be performed using either of the following options of SQL*DBA: the Start Up Instance dialog box with the Mount radio button selected or the STARTUP command with the MOUNT option.
8. If one or more damaged data files were restored to alternative locations in Step 5, the relocation of these files must be indicated to the control file of the associated database. Therefore, use the operation discussed in “Renaming and Relocating Data Files” on page 7-13, as necessary.
9. All data files of the database must be online during time-based or change-based recovery (unless an offline tablespace was taken offline normally). You can determine the names of all data files to be recovered using the list of data files that accompanies the control file being used (if available; see “Listing Database Files Before Backup” on page 18-6) or query the V\$DATAFILE view. Then, use the ALTER DATABASE command and the DATAFILE ONLINE option to make sure that all data files of the database are online. For example, to guarantee that a data file named USERS1 (a fully specified filename) is online, enter the following statement:

```
ALTER DATABASE DATAFILE 'users1' ONLINE;
```

If a data file is specified that is already online, ORACLE ignores the statement.
10. **Depending on the type of incomplete media recovery being performed, follow either Step 10a, 10b, or 10c.**
 - a. If a backup of the control file is being used with this incomplete recovery (that is, a control file backup or re-created control file was restored in Step 4), indicate this in the dialog box or command used to start recovery (that is, specify USING BACKUP CONTROLFILE).

- b. Use either of the following options of SQL*DBA to begin cancel-based recovery: the Recover Closed Database dialog box with the Until User Cancel radio button selected or an equivalent RECOVER DATABASE UNTIL CANCEL statement.
 - c. Use either of the following options of SQL*DBA to begin time-based recovery: the Recover Closed Database dialog box with the Until Time radio button selected and specified end time, or an equivalent RECOVER DATABASE UNTIL TIME statement. In either case, the time is always specified using the following format, delimited by single quotes, 'YYYY-MM-DD:HH.24:MI:SS'.
 - d. Use either of the following options of SQL*DBA to begin change-based recovery: the Recover Closed Database dialog box with the Until Change radio button selected and specified last change number, or an equivalent RECOVER DATABASE UNTIL CHANGE statement. In either case, the change number (SCN) is specified as a decimal number without quotes.
11. ORACLE now begins the roll forward phase of media recovery by applying the necessary redo log files (archived and online) to reconstruct the restored data files. If automated, the suggested log files are automatically applied; if not automated, ORACLE prompts for the required redo log file. See "Applying Redo Log Files" on page 19-12 for more information.
- ORACLE continues to apply redo log files. Depending on the type of incomplete recovery being performed, follow either of the next two steps:
- a. If performing cancel-based recovery, continue applying redo log files until the most recent, undamaged redo log file has been applied to the restored data files.
 - b. If performing time-based or change-based recovery, continue applying redo log files until the last required redo log file has been applied to the restored data files.
12. Depending on the type of incomplete media recovery being used, follow either Step 12a or 12b:
- a. If performing cancel-based recovery, enter "CANCEL" to cancel recovery after ORACLE has applied the redo log file just prior to the damaged file. Cancel-based recovery is now complete.
 - b. If performing time-based or change-based recovery, ORACLE automatically terminates the recovery operation when it has reached the correct time or change.

In either case, a message is returned to indicate the result of the incomplete recovery. If recovery was indeed incomplete (that is, some redo information was not applied), the following message is reported in the ALERT file:

Incomplete recovery done UNTIL CHANGE *scn*

However, if the recovery operation was complete (that is, all available redo was applied), the following messages are reported in the ALERT file:

Incomplete recovery applied all redo ever generated.
Recovery completed through change *nnnnnnnn*

13. The first time you open the database subsequent to incomplete media recovery, you must explicitly specify whether or not to reset the log sequence number, by including either the RESETLOGS or NORESETLOGS option. Resetting the redo log discards any redo information that was not applied during recovery, ensuring that it will never be applied; it also reinitializes the control file information about online redo logs and redo threads, clears the contents of the online redo logs, creates the online redo log files if they do not currently exist, and resets the log sequence number to 1. Use the following rules to decide whether to specify RESETLOGS or NORESETLOGS:

- Reset the log sequence number if you used a backup of the control file in recovery, no matter what type of recovery was performed (complete or incomplete).
- Reset the log sequence number if the recovery was actually incomplete (see message in Step 12). For example, you must have specified a previous time or SCN, not one in the future.
- Do not reset logs if recovery was complete (unless you used a backup control file). This applies both when you intentionally performed complete recovery and when you performed incomplete recovery but actually recovered all changes in the redo logs anyway. (See the explanation in step 12 for how to examine the ALERT file to see if incomplete recovery was actually complete.)

To preserve the log sequence number when opening a database after recovery, use the SQL command ALTER DATABASE with the OPEN NORESETLOGS option. To reset the log sequence number when opening a database after recovery, use the SQL command , ALTER DATABASE with the OPEN RESETLOGS option. (If you attempt to reset the log when you should not, or if you neglect to

reset the log when you should, ORACLE returns an error and does not open the database. Correct the error and try again.)

If the log sequence number is reset when opening a database, different messages are returned, depending on whether the recovery was complete or incomplete. If the recovery was complete, the following message is reported in the ALERT file:

RESETLOGS after complete recovery through change *scn*

If the recovery was incomplete, the following message is reported in the ALERT file:

RESETLOGS after incomplete recovery UNTIL CHANGE *scn*

14. If you reset the redo log sequence when opening the database, immediately shut down the database normally and make a full database backup. (See “Performing Full Backups” on page 18-7.) Otherwise, you will not be able to recover changes made after you reset the logs; until you take a full backup, the only way to recover will be to repeat the procedures you just finished, up to resetting the logs. (You do not need to back up the database if you did not reset the log sequence.)

Examples of Media Failures and Appropriate Recovery Procedures

The following sections describe common media failures, the effects of a media failure, and the appropriate steps necessary to recover a database in each situation.

Types of Media Failures

Media failures can be separated into two general categories: permanent and temporary media failures. Permanent media failures are serious hardware problems that cause the permanent loss of data on the disk. Lost data cannot be recovered except by repairing or replacing the failed storage device and restoring backups of the files stored on the damaged storage device. Temporary media failures are hardware problems that do not detrimentally affect the storage of data. The following are two examples of situations that can be categorized as temporary media failures:

- A disk’s controller fails. Once the disk’s controller is replaced, the data on the disk can be accessed.
- Power to a storage device is cut off. Once the power is returned, the storage device and all associated data is accessible, as normal.

Loss of Data Files

If a media failure has affected data files of a database, the appropriate recovery procedure depends on the archiving mode of the database, the type of media failure, and the exact files affected by the media failure. The following sections explain the appropriate recovery strategy in various situations.

Loss of Data Files, NOARCHIVELOG Mode

If either a permanent or temporary media failure affects **any** data files of a database operating in NOARCHIVELOG mode, ORACLE automatically shuts down the database. Depending on the type of media failure, one of two recovery paths can be used:

- If the media failure is temporary, correct the temporary hardware problem and restart the database. In most cases, instance recovery is possible and all committed transactions can be recovered using the online redo log.
- If the media failure is permanent, follow the steps in “Restoring a Full Backup” on page 19-16 to recover from the media failure.

Loss of Data Files, ARCHIVELOG Mode

If either a permanent or temporary media failure affects the data files of a database operating in ARCHIVELOG mode, the following situations can exist.

- If a temporary or permanent media failure affects any data files of the SYSTEM tablespace or any data files that contain active rollback segments, the database becomes inoperable and should be immediately shut down, if it has not already been shut down by ORACLE.

If the hardware problem is temporary, correct the problem and restart the database. In most cases, instance recovery is possible and all committed transactions can be recovered using the online redo log.

If the hardware problem is permanent, follow the procedure given in “Performing Closed Database Recovery” on page 19-18.

- If a temporary or permanent media failure affects only data files not mentioned in the previous item, the affected data files are unavailable, but the database can continue to operate.

If the unaffected portions of the database must remain available, do not shut down the database. First, take all tablespaces that contain problem data files offline using the temporary option. Next, follow the procedure in “Performing Open Database-Offline Tablespace Recovery” on page 19-20. **If the problem that caused the failure is temporary, do not follow Step 3;** recovery can proceed using the intact data files, and often only requires only the online redo log files.

Loss of Online Redo Log Files

If a media failure has affected the online redo log of a database, the appropriate recovery procedure depends on the configuration of the online redo log (mirrored or non-mirrored), the type of media failure (temporary or permanent), and the types of online redo log files affected by the media failure (current, active, not yet archived, or inactive online redo log files). The following sections explain the appropriate recovery strategies in various situations.

Loss of Online Redo Log Members or Mirrored Online Redo Log

If the online redo log of a database is mirrored and at least one member of each online redo log group is not affected by the media failure, ORACLE allows the database to continue functioning as normal (error messages are written to the LGWR trace file and ALERT file of the database). However, the problem should be handled using one of the following actions:

- If the hardware problem is temporary, correct the problem. After it has been fixed, LGWR accesses the previously unavailable online redo log files as if a problem never happened.
- If the hardware problem is permanent, use the following procedure:
 1. Make sure that there are at least two members in each online redo log group, to safeguard against a single point of failure. If any online redo log group contains only one member as a result of a media failure, create additional online redo log files for the group.
 2. Drop the damaged online redo log files.
 3. Repair or replace the damaged storage device, if desired. After you complete this step, you can re-create the damaged set of online redo log files and drop the set of online redo log files made in Step 1 of this procedure.

Note: If all members of an online redo log group in a mirrored online redo log are damaged by a media failure, see the following section.

Loss of All Online Redo Log Members of an Online Redo Log Group

If all members of an online redo log group are damaged by a media failure, different situations can occur, depending on the type of online redo log group affected by the failure and the archiving mode of the database. To determine in which state the damaged online redo log group was, query the view V\$LOG:

```
SELECT group#, members, status, archived
FROM v$log
;
```

GROUP#	MEMBERS	STATUS	ARCHIVED
0001	log1a	INACTIVE	YES
0001	log1b	INACTIVE	YES
0002	log2a	ACTIVE	YES
0002	log2b	ACTIVE	YES
0003	log3a	CURRENT	NO
0003	log3b	CURRENT	NO

The following sections explain the various situations:

Loss of an Inactive, Archived Online Redo Log Group If all members of an inactive (but archived, if you are in ARCHIVELOG mode) online redo log group are damaged, several situations can arise:

- In either archiving mode, if a temporary media failure affects only an inactive online redo log group, correct the problem; LGWR can reuse the group when required.
- If a media failure permanently prevents access to only an inactive online redo log group, the damaged inactive online redo log group will eventually halt normal database operation.

If you notice the problem before the database halts, either drop the damaged online redo log group (adding another group if necessary) or, if you are in ARCHIVELOG mode, restore the archive of the online redo log from offline storage, creating enough members for the group and renaming the online redo log to the new location.

If the database has already halted, use the following steps to resolve this problem:

1. Abort the current instance immediately using either the Abort Instance option of SQL*DBA'S Shut Down menu or the SHUTDOWN command with the ABORT option.
2. Start a new instance and mount the database, but do not open the database. This operation can be performed using either of the following options of SQL*DBA: the Start Up Instance dialog box with the Mount radio button selected or the STARTUP command with the MOUNT option.
3. If you are in ARCHIVELOG mode, alter the database's archiving mode by using the ALTER DATABASE command with the NOARCHIVELOG option:

```
ALTER DATABASE NOARCHIVELOG;
```
4. Create a new online redo log group to replace the damaged online redo log group using either the Add Online Redo Log Group dialog box of SQL*DBA or the SQL command ALTER DATABASE with the ADD LOGFILE parameter, or restore an archived copy of the damaged redo log group from disk.
5. Drop the damaged online redo log group using either the Drop online Redo Log Group dialog box of SQL*DBA or the SQL command ALTER DATABASE with the DROP LOGFILE parameter.
6. If you were previously in ARCHIVELOG mode, switch the database's archiving mode back to ARCHIVELOG:

```
ALTER DATABASE ARCHIVELOG;
```
7. Immediately back up the database; also backup the database's control file using the ALTER DATABASE command with the BACKUP CONTROLFILE option.

You can now open the database.

Loss of an Unarchived Online Redo Log Group In ARCHIVELOG mode only, if an online redo log group that has not yet been archived is damaged, several situations can arise:

- If the media failure is temporary, correct the problem; LGWR can archive and reuse the group when required.
- If the media failure permanently prevents access to the unarchived redo log, the damaged unarchived online redo log group will eventually halt normal database operation when the

ARCH process cannot archive the log and eventually all logs are waiting to be archived. Use the following steps to repair this problem:

1. Shut down the database normally, using either the Normal option of SQL*DBA's Shut Down dialog or the SHUTDOWN command with the NORMAL option. If you cannot shutdown the database normally, use either the IMMEDIATE option or the ABORT option, then restart the instance using the RECOVER option, mount the database, and shut it down normally.
2. Start cancel-based recovery, canceling immediately, before you apply any redo logs.
3. Rename the damaged online redo log group to a new location. (You do not need to create the files.)
4. Open the database using the RESETLOGS option.
5. Immediately backup the control file of the database using the ALTER DATABASE command with the BACKUP CONTROLFILE parameter.

The damaged log is now repaired.

Loss of an Active Online Redo Log Group If all members of an active online redo log group are damaged, several situations can arise:

- If the media failure is temporary, correct the problem and ORACLE can reuse the group when required.
- If the database is operating in NOARCHIVELOG mode and a permanent media failure permanently prevents access to an active online redo log group, recover the database from a full backup.
- If the database is operating in ARCHIVELOG mode and a permanent media failure permanently prevents access to an active online redo log group, shut down the database normally (if it has not already crashed); how you recover depends on whether the online redo log has been archived or not, as explained below.

If the active redo log has already been archived, restore the group from the archived copy, renaming it to the restored location, and restart the database. ORACLE can continue to use the group as before.

If the active redo log has not yet been archived, use the procedure given in “Performing Cancel-Based, Time-Based, or Change-Based Recovery” on page 19-23, recovering up through the log before the damaged log, and modifying the following steps:

8. Rename the damaged online redo log group to a new location. (You do not need to create the files.)
14. In addition to making a full backup, immediately backup the control file of the database using the ALTER DATABASE command with the BACKUP CONTROLFILE parameter.

The damaged log is now repaired.

Loss of the Current Online Redo Log Group If all members of the current online redo log group are damaged, several situations can arise:

- If the media failure is temporary, correct the problem and ORACLE can reuse the group when required.
 - If the database is operating in NOARCHIVELOG mode and a permanent media failure permanently prevents access to an active online redo log group, recover the database from a full backup.
 - If the database is operating in ARCHIVELOG mode and a permanent media failure permanently prevents access to an active online redo log group, the database will crash. Use the procedure given in “Performing Cancel-Based, Time-Based, or Change-Based Recovery” on page 19-23, recovering up through the log before the damaged log and modifying the following steps:
8. Rename the damaged online redo log group to a new location. (You do not need to create the files.)
 14. In addition to making a full backup, immediately backup the control file of the database using the ALTER DATABASE command with the BACKUP CONTROLFILE parameter.

The damaged log is now repaired.

Loss of Multiple Redo Log Groups If you have lost multiple groups of the online redo log, use the recovery method for the most difficult log to recover. The order of difficulty, from most difficult to least, is as follows: the current online redo log, an active online redo log, an unarchived redo log, an inactive online redo log.

Loss of Archived Redo Log Files

If the database is operating so that filled online redo log groups are being archived and the only copy of an archived redo log file is damaged, it does not affect the present operation of the database. However, the following situations can arise if media recovery is required in the future:

- If **all** data files have been backed up after the filled online redo log group was written (which is now archived), the archived version of the filled online redo log group is not required for any complete media recovery operation.
- Assume the most recent backup file of a data file was taken after the filled online redo log group was written (the group now corresponds to the damaged archived redo log file). At some future point, the corresponding data file is damaged by a permanent media failure. The most recent backup of the damaged data file must be used and incomplete media recovery can only recover the database up to the damaged archived redo log file.
- If time-based recovery is needed, the damaged archived redo log file may be required if old data file backups are used that were taken before the original online redo log group was written. In this case, the incomplete media recovery can only recover the database up to the damaged archived redo log group.



If you know that an archived redo log group has been damaged, immediately backup all data files so that you will have a complete backup that does not require the damaged archived redo log.

Loss of Control Files

If a media failure has affected the control files of a database (whether control files are mirrored or not), the database continues to run until the first time that ORACLE needs to access the control files; at this point, the database and instance are automatically shut down.

If the media failure is temporary and the database has not yet shut down, immediately correcting the media failure can avoid the automatic shut down of the database. However, if the database does shut down before the temporary media failure is corrected, you can simply restart the database after fixing the problem (and restoring access to the control files).

The appropriate recovery procedure for media failures that permanently prevent access to control files of a database depends on whether you have mirrored the control files. The following sections detail the appropriate procedures to follow.

Loss of Mirrored Control Files

Use the following steps to recover a database after one or more control files of a database have been damaged by a permanent media failure and at least one control file has not been damaged by the media failure:

Note: If all control files of a mirrored control file configuration have been damaged, follow the instructions for recovering from the loss of non-mirrored control files.

1. If the instance is still running, immediately abort the current instance using either of the following options of SQL*DBA: the Abort Instance option of the Shut Down menu or the SHUTDOWN command with the ABORT option.
2. Correct the hardware problem that caused the media failure. If the hardware problem cannot be repaired quickly, database recovery can proceed by restoring damaged control files to an alternative storage device; proceed to Step 3.
3. Use an intact copy of the database's control file to copy over the damaged control files. If possible, copy the intact control file to the original locations of all damaged control files; if the hardware problem persists, copy the intact control file to alternate locations. If you restored all damaged control files to their original location, proceed to Step 5; if all damaged control files were not restored, or not restored to their original location, proceed to Step 4.
4. If all damaged control files were not restored, or not restored to their original location in Step 3, the parameter file of the database must be edited so that the CONTROL_FILES parameter reflects the current locations of all control files and does not include any control files that are not present.

You can now start a new instance, and mount and open the database.

Loss of Non-Mirrored Control Files

If all control files of a database have been damaged by a permanent media failure, complete media recovery can be performed using a backup of the control file.

Note: If a backup of the control file is not available, you must create a new control file for the database; see "Creating a New Control File" on page 6-4.

If a control file backup is available, use the procedure given in "Performing Closed Database Recovery" on page 19-18 using the following alternative steps when required:

- Between Steps 2 and 3, restore a backup of the control file that reflects the current structure of the database. If the control file is restored to an alternate location, also edit the `CONTROL_FILES` parameter in the database's parameter file before starting an instance in Step 5 (on page 19-18).
- If no data files are damaged, Steps 3, 6, and 7 can be skipped.
- Choose Step 8a, rather than 8b. Include the `USING BACKUP CONTROLFILE` option. (This option tells the ORACLE instance that it will detect inconsistencies between the control file and the data and redo file headers; the option is needed any time the control file used in recovery is not current.)
- When you are ready to open the database, first take a full offline backup; this is a precaution in case a problem arises in opening the database with the `RESETLOGS` option. Then, open the database using the `RESETLOGS` option. (The redo log sequence number is reset to 1.) Immediately after opening the database, because the log sequence has been reset and all old redo information discarded, make a full backup of the database.

Recovery From User Errors

Serious user errors can detrimentally affect a database and necessitate database recovery. Recovering a database from a user error can be performed so that virtually no data is lost using the following steps:

Note: If the database administrator has properly granted powerful privileges (such as `DROP ANY TABLE`) to only selected, appropriate users, user errors that require database recovery are minimized.

1. Backup the existing, intact database.
2. Leave the existing database intact, but reconstruct a temporary copy of the database up to the user error using time-based recovery.
3. Export the data lost to the user error from the reconstructed, temporary copy of the database.
4. Import the data lost by the user error into the permanent database.
5. Delete the files associated with the temporary copy of the reconstructed database to conserve disk space.

For example, the steps below recover a table that was accidentally dropped:

1. The database that experienced the user error can remain online and available for normal use. The database can remain open or be shut down. Back up all data files of the existing database in case an error is made during the remaining steps of this procedure.

2. Create a temporary copy of the database to a past point in time using time-based recovery. Use the procedure in “Performing Cancel-Based, Time-Based, or Change-Based Recovery” on page 19-23; however, do not cause a conflict with the existing control file of the permanent database. Restore a single control file backup to an alternative location in Step 4 and edit the parameter file, as necessary, or create a new control file at the alternative location (changing the database name, if you like). Also, restore all data files to alternative locations in Step 5 so that you do not affect the permanent copy of the database.



Utilities



Utilities

3. Export the lost data using Export from the temporary, restored version of the database. In this case, export the accidentally dropped table. (See the *ORACLE7 Server Utilities User's Guide*.)
4. Import the data exported in Step 3 into the permanent copy of the database using the ORACLE utility Import. (See the *ORACLE7 Server Utilities User's Guide*.)
5. Delete the files of the temporary, reconstructed copy of the database to conserve disk space.

PART

VII

DATABASE AND INSTANCE TUNING

UNDERSTANDING THE TUNING PROCESS

ORACLE is a sophisticated and highly tunable software product. Its flexibility allows you to make small adjustments that affect performance. By tuning ORACLE, you can tailor its performance to best meet your needs.

In this chapter, you will establish your goals for the tuning process. After establishing these goals, you can more easily choose the tuning measures in this Guide that will most benefit you.

This chapter also presents an overview of the tuning method recommended by Oracle. This method is a complete, step by step process for tuning ORACLE. The overview discusses the scope, importance, and benefits of each step of the method. If you have never tuned ORACLE, you should follow these steps in order. This method provides you with an approach to tuning that is both sound and straightforward. It also introduces you to the performance diagnostic tools provided by ORACLE. If you are already familiar with tuning ORACLE or if you think you know which part of your database needs tuning, this chapter helps you find the specific information you need.

This chapter also lists some of the most important individual tips to consider when tuning ORACLE. These tips highlight the tuning process.

Setting Goals for Tuning

Before you begin tuning ORACLE, you should establish goals for tuning. These goals should stem directly from your reasons for tuning and should be based on the business requirements for which you use ORACLE. For example, you may want to tune ORACLE so that your order entry application can accept a high volume of orders during the peak of the business day.

Your goals for tuning may include one or more of these goals:

- to improve the performance of a specific type of SQL statement
- to improve the performance of a specific database application
- to improve the overall performance of all concurrent users and applications on your database

Keep your goals in mind as you consider each of the tuning measures in this Guide. To decide whether to implement a particular measure, you should consider its performance benefits in light of your goals.

Also keep in mind that your goals may conflict. For example, to achieve best performance for a specific SQL statement, you may have to sacrifice the performance of other SQL statements running concurrently on your database.

Steps of the Tuning Process

The process of tuning ORACLE has three steps. Each step is described in one of the following chapters:

Step One	Chapter 21 “Tuning Memory Allocation” of this Guide
Step Two	Chapter 22 “Tuning I/O” of this Guide
Step Three	Chapter 23 “Tuning Contention” of this Guide

These steps are designed to be as modular and independent as possible. For example, the benefits of tuning I/O in Step Two are separate from the benefits of eliminating contention in Step Three.

However, decisions you make in one step may influence subsequent steps. For example, disk I/O, which is tuned in Step Two, depends on the size of the buffer cache, which is tuned in Step One. The steps that have the greatest effects on other steps as well as the greatest effect on performance appear early in the method. For this reason, you should follow these steps in order.

Before Beginning the Process



AppDev

Good performance begins with good application design and efficient SQL statements. Before beginning the process of tuning ORACLE itself, you should be certain that your applications have been designed to take full advantage of ORACLE features and that your SQL statements are written to be most efficiently processed by ORACLE. Guidelines for designing applications and writing SQL statements for best performance appear in Chapter 1 “The Application Developer” and Chapter 5 “Tuning SQL Statements” of the *ORACLE7 Server Application Developer’s Guide*, respectively. Discuss these chapters with the developers of your applications.

Step One: Tuning Memory Allocation

Appropriate allocation of memory resources to ORACLE memory structures can have a large impact on performance. In Step One, you will decide how much memory to allocate to the following structures:

- SQL and PL/SQL areas
- the data dictionary cache
- the buffer cache

Proper allocation of memory resources can yield these benefits:

- improved cache performance
- reduced parsing of SQL statements and PL/SQL blocks
- reduced paging and swapping

Step Two Tuning I/O

Disk I/O tends to reduce the performance of many software applications. However, ORACLE is designed so that its performance need not be limited by I/O.

Tuning I/O involves these procedures:

- distributing I/O to avoid disk contention
- storing data in data blocks for best access
- creating extents large enough for your data

Step Three: Tuning Contention

Concurrent processing by multiple ORACLE users may create contention for ORACLE resources. Contention may cause processes to wait until resources are available. In Step Three you will reduce contention for:

- rollback segments
- processes of the multi-threaded server architecture
- redo log buffer latches

After Completing the Process

After completing Step Three, reassess performance and decide whether further tuning is necessary. Since some performance gains made in later steps may pave the way for further improvements in earlier steps, additional passes through the tuning process may be useful.

You should also consider the tuning measures discussed in Chapter 24 “Additional Tuning Considerations” of this Guide. These measures tune ORACLE operations that reduce performance only in special cases.

Top Tuning Tips

Some parts of the tuning process are more important than others. The following list highlights the most important parts of tuning. This list is by no means an exhaustive collection of all the material presented in this Guide.

Pay particular attention to the issues addressed in this list. If you have not tuned ORACLE before, this list gives you perspective of the relative importance of the parts of the tuning process. If you are familiar with tuning ORACLE, this list directs you to the parts of this Guide and other ORACLE manuals that discuss these issues.



1. Index tables appropriately. Proper use of indexes may improve the performance of some SQL statements by orders of magnitude. For information on indexes, see Chapter 5 “Tuning SQL Statements” of the *ORACLE7 Server Application Developer’s Guide*.



2. Use cost-based optimization whenever possible. With this approach, the optimizer uses statistics for tables, indexes, and clusters to choose the fastest way to execute a SQL statement. This approach reduces the time and effort necessary for you to tune your SQL statements. For information on how to use the cost-based approach, see Chapter 5 “Tuning SQL Statements” of the *ORACLE7 Server Application Developer’s Guide*.
3. To direct the optimizer to choose a specific way to execute a SQL statement, use *hints*, or suggestions within a comment within the statement. For information on hints, see Chapter 5 “Tuning SQL Statements” of the *ORACLE7 Server Application Developer’s Guide*.



4. Take advantage of other new features of ORACLE7. Enforced integrity constraints are discussed in Chapter 5 “Maintaining Data Integrity” of the *ORACLE7 Server Application Developer’s Guide*. Stored Procedures are discussed in Chapter 7 “Using Procedures and Packages” of the *ORACLE7 Server Application Developer’s Guide*. Shared SQL and PL/SQL areas are discussed in Chapter 21 “Tuning Memory Allocation” of this Guide.
5. Reduce the time spent parsing by keeping frequently-used SQL statements and stored procedures parsed after execution and by keeping dictionary data in memory. For information on these measures, see Chapter 21 “Tuning Memory Allocation” of this Guide.
6. If excessive I/O of data blocks occurs, increase the size of the buffer cache in the System Global Area (SGA). For information on resizing ORACLE memory structures, see Chapter 21 “Tuning Memory Allocation” of this Guide.
7. On operating systems with virtual storage, be sure the System Global Area (SGA) fits into real memory to ensure fast access to cached data. For information on allocating memory, see Chapter 21 “Tuning Memory Allocation” of this Guide.
8. Distribute I/O so that none of the disks containing your data files and redo log files are operating near full capacity. For information on distributing I/O, see Chapter 22 “Tuning I/O” of this Guide.
9. Reduce contention for latches if the percentage of latch requests that result in waiting significantly exceeds 1%. For methods of monitoring and reducing latch contention, see Chapter 23 “Tuning Contention” of this Guide.
10. Be sure all ORACLE processes, both foreground and background, have the same operating system process priority. Performance problems may occur if your operating system swaps out a low priority process while that process has exclusive access to a resource. A high priority process in need of that resource may wait indefinitely for the low priority process to be swapped back in. For information on problems caused by contention, see Chapter 23 “Tuning Contention” of this Guide.

TUNING MEMORY ALLOCATION

This chapter presents Step One of the tuning process: tuning memory allocation. This chapter discusses tuning the performance of the following ORACLE memory structures:

- private SQL and PL/SQL areas
- the shared pool
- the buffer cache

The performance of ORACLE can be improved by proper sizing of these structures. In this chapter, you will allocate memory to these structures to achieve best performance.

The Importance of Memory Allocation

ORACLE stores information in two places:

- in memory
- on disk

Since memory access is much faster than disk access, it is desirable for data requests to be satisfied by access to memory rather than access to disk. For best performance, it is advantageous to store as much data as possible in memory rather than on disk. However, memory resources on your operating system are likely to be limited. Tuning memory allocation involves distributing available memory to ORACLE memory structures.

Because ORACLE'S memory requirements vary depending on your application, you should tune memory allocation after tuning your application and your SQL statements based on the recommendations presented in Chapter 1 "The Application Developer" and Chapter 5 "Tuning SQL Statements" of the *ORACLE7 Server Application*



AppDev

Developer's Guide. Allocating memory before tuning your application and your SQL statements may make it necessary to resize some ORACLE memory structures to meet the needs of your modified statements and application.

Also, you should tune memory allocation before considering the information in Chapter 22 "Tuning I/O" of this Guide. Allocating memory establishes the amount of I/O necessary for ORACLE to operate. This chapter shows you how to allocate memory in order to perform as little I/O as possible. Chapter 22 shows you how to perform that I/O as efficiently as possible.

Steps for Tuning Memory Allocation

This section outlines the process of tuning memory allocation. For best results, you should follow these steps in the order they are presented. Each step is described in more detail in the remainder of this chapter.

Tuning Your Operating System

Ensuring that your operating system runs smoothly and efficiently establishes a solid basis for allocating memory to ORACLE. This step also gives you a good idea of the amount of memory on your operating system that is available for ORACLE.

Tuning Private SQL and PL/SQL Areas

The use of SQL and PL/SQL areas and the frequency of parse calls are primarily determined by your application. Since parsing affects the frequency of access to the data dictionary, you should tune private SQL and PL/SQL areas before tuning the data dictionary cache in the shared pool.

Tuning the Shared Pool

Tuning the shared pool involves allocating memory for these memory structures:

- the Library cache containing shared SQL and PL/SQL areas
- the data dictionary cache
- information for sessions connected through shared server processes

A cache miss on the data dictionary cache miss or library cache is more expensive than a miss on the buffer cache. For this reason, you should allocate sufficient memory for the shared pool first.

Tuning the Buffer Cache

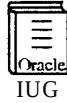
After tuning private SQL and PL/SQL areas and the shared pool, you can devote the remaining available memory to the buffer cache.

It maybe necessary to repeat the steps of memory allocation after the initial pass through the process. Subsequent passes allow you to make adjustments in earlier steps based on changes in later steps. For example, if you increase the size of the buffer cache, you may need to allocate more memory to ORACLE to avoid paging and swapping.

Tuning Your Operating System

You should begin tuning memory allocation by tuning your operating system with these goals:

- to reduce paging and swapping
- to fit the System Global Area (SGA) into main memory
- to allocate enough memory to individual users



These goals apply in general to most operating systems. However, the details of tuning your operating system vary depending on which operating system you are using.

Reducing Paging and Swapping

Your operating system may store information in any of these places:

- real memory
- virtual memory
- expanded storage
- disk

Your operating system may also move information from one storage location to another. Depending on your operating system, this movement is called paging or swapping. Many operating systems page and swap to accommodate large amounts of information that do not fit into real memory. However, paging and swapping take time. Excessive paging or swapping can reduce the performance of many operating systems.

Monitor your operating system behavior with operating system utilities. Excessive paging or swapping indicates that new information is often being moved into memory. In this case, your system's total memory may not be large enough to hold everything for which you have allocated memory. You should either increase the total memory on your system or decrease the amount of memory you have allocated.

Tuning the System Global Area (SGA)

Since the purpose of the System Global Area (SGA) is to store data in memory for fast access, the SGA should always be contained in main memory. If the SGA is swapped to disk, its data is no longer so quickly accessible. On most operating systems, the disadvantage of excessive paging significantly outweighs the advantage of a large SGA, so you should ensure that the entire SGA always fits into memory and is not paged or swapped.

You can see how much memory is allocated to the SGA and each of its internal structures by issuing this SQL*DBA statement:

```
SQLDBA> SHOW SGA
```

The output of this statement might look like this:

Total System Global Area	3554188	bytes
Fixed Size	22298	bytes
Variable Size	3376332	bytes
Database Buffers	122880	bytes
Redo Buffers	32768	bytes

Some operating systems for IBM mainframe computers are equipped with *expanded storage*, or special memory in addition to main memory to which paging can be performed very quickly. These operating systems may be able to page data between main memory and expanded storage faster than ORACLE can read and write data between the SGA and disk. For this reason, allowing a larger SGA to be swapped may lead to better performance than ensuring that a smaller SGA stays in main memory. If your operating system has expanded storage, you can take advantage of it by allocating a larger SGA despite the resulting paging.

User Memory Allocation

On some operating systems, you may have control over the amount of physical memory allocated to each user. Be sure all users are allocated enough memory to accommodate the resources they need to use their application with ORACLE. Depending on your operating system, these resources may include:

- the ORACLE executable image
- the SGA
- ORACLE application tools
- application-specific data

On some operating systems, ORACLE software can be installed so that a single executable image can be shared by many users. By sharing executable images among users, you can reduce the amount of memory required by each user.

Tuning Private SQL and PL/SQL Areas

In this section you learn how to tune private SQL and PL/SQL areas. Tuning private SQL areas involves identifying unnecessary parse calls made by your application and then reducing them. To reduce parse calls, you may have to increase the number of private SQL areas that your application can have allocated at once. Throughout this section, information about private SQL areas and SQL statements also applies to private PL/SQL areas and PL/SQL blocks.

Identifying Unnecessary Parse Calls

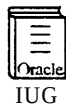
To identify unnecessary parse calls, run your application with the SQL trace facility enabled. For each SQL statement in the trace output, examine the *count* statistic for the Parse step. This statistic tells you how many times your application makes a parse call for the statement. This statistic includes parse calls that are satisfied by access to the library cache as well as parse calls that result in actually parsing the statement. Note that this statistic does not include implicit parsing that occurs when an application executes a statement whose shared SQL area is no longer in the library cache. For information on detecting implicit parsing, see the section “Examining Library Cache Activity” on page 21-8.

If the *count* value for the Parse step is near the *count* value for the Execute step for a statement, your application maybe deliberately making a parse call each time it executes the statement. Try to reduce these parse calls through your application tool.

Reducing Unnecessary Parse Calls

Depending on the ORACLE application tool you are using, you maybe able to control how frequently your application performs parse calls and allocates and deallocates private SQL areas. Whether your application reuses private SQL areas for multiple SQL statements determines how many parse calls your application performs and how many private SQL areas the application requires. In general, an application that reuses private SQL areas for multiple SQL statements does not need as many private SQL areas as an application that does not. However, an application that reuses private SQL areas must perform more parse calls because the application must make a new parse call whenever an existing private SQL is reused for a new SQL statement.

Be sure that your application can open enough private SQL areas to accommodate all of your SQL statements. If you allocate more private SQL areas, you may need to increase the limit on the number of cursors permitted for a session. You can increase this limit by increasing the value of the initialization parameter `OPEN_CURSORS`. The maximum value for this parameter depends on your operating system. The minimum value is 5.



The means by which you control parse calls and allocation and deallocation of private SQL areas varies depending on your ORACLE application tool. The following sections introduce the means used for some tools. Note that these means apply only to private SQL areas and not to shared SQL areas.

Reducing Parse Calls with the ORACLE Precompilers With the ORACLE Precompilers, you control private SQL areas and parse calls with these options:

- HOLD_CURSOR
- RELEASE_CURSOR
- MAXOPENCURSORS

These options can be specified in two ways:

- on the precompiled command line
- within the precompiled program



Precomp

With these options, you can employ different strategies for managing private SQL areas during the course of the program. For information on these options, see the *Programmer's Guide to the ORACLE Precompilers* or the *Programmer's Guide to the Pro*Ada Precompiler*.

Reducing Parse Calls with the ORACLE Call Interfaces (OCIS)

With the ORACLE Call Interfaces (OCIS), you have complete control over parse calls and private SQL areas with these OCI calls:

OSQL3	An OSQL3 or OPARSE call allocates a private SQL
OPARSE	area for a SQL statement.
OCLOSE	An OCLOSE call closes a cursor and deallocates the private SQL area of its associated statement.



OCI

For more information on these calls, see the *Programmer's Guide to the ORACLE Call Interfaces* or the *Pro*Ada ORACLE Call Interface User's Guide*.

Reducing Parse Calls with SQL*Forms With SQL*Forms, you also have some control over whether your application reuses private SQL areas. You can exercise this control in three places:

- at the trigger level
- at the form level
- at runtime

For more information on the reuse of private SQL areas by SQL*Forms, see the *SQL*Forms Designer's Reference* manual.

Tuning the Shared Pool

This section describes tuning these parts of the shared pool:

- library cache
- data dictionary cache
- session information

This section presents these parts of the shared pool in order of importance. Because the algorithm that ORACLE uses to manage data in the shared pool that tends to hold dictionary data in memory longer than library cache data, tuning the library cache to an acceptable cache hit ratio often ensures that the data dictionary cache hit ratio is also acceptable. Allocating space in the shared pool for session information is only necessary if you are using the multi-threaded server architecture.

Tuning the Library Cache

The library cache contains shared SQL and PL/SQL areas. This section tells you how to tune the library cache by:

- examining library cache activity
- reducing library cache misses
- speeding access to shared SQL and PL/SQL areas in the library cache

Throughout this section, information about shared SQL areas and SQL statements also applies to shared PL/SQL areas and PL/SQL blocks.

Examining Library Cache Activity

Library cache misses can occur on either of these steps in the processing of a SQL statement.

Parse If an application makes a parse call for a SQL statement and the parsed representation of the statement does not already exist in a shared SQL area in the library cache, ORACLE parses the statement and allocates a shared SQL area. You may be able to reduce library cache misses on parse calls by ensuring that SQL statements can share a shared SQL area whenever possible.

Execute If an application makes an execute call for a SQL statement and the shared SQL area containing the parsed representation of the statement has been deallocated from the library cache to make room for another statement, ORACLE implicitly reparses the statement, allocates a new shared SQL area for it, and executes it. You may be able to reduce library cache misses on execution calls by allocating more memory to the library cache.

Determine whether misses on the library cache are affecting the performance of ORACLE by querying the dynamic performance table V\$LIBRARYCACHE.

The V\$LIBRARYCACHE Table Statistics reflecting library cache activity are kept in the dynamic performance table V\$LIBRARYCACHE. These statistics reflect all library cache activity since the most recent instance startup. To monitor library cache activity, examine this table. By default, this table is only available to the user SYS and to users granted SELECT ANY TABLE system privilege, such as SYSTEM.

Each row in this table contains statistics for one type of item kept in the library cache. The item described by each row is identified by the value of the NAMESPACE column. Rows of the table with these NAMESPACE values reflect library cache activity for SQL statements and PL/SQL blocks:

- 'SQL AREA'
- 'TABLE / PROCEDURE'
- 'BODY'
- 'TRIGGER'

Rows with other NAMESPACE values reflect library cache activity for object definitions that ORACLE uses for dependency maintenance.

These columns of the V\$LIBRARYCACHE table reflect library cache misses on execution calls:

PINS	This column shows the number of times an item in the library cache was executed.
RELOADS	This column shows the number of library cache misses on execution steps.

Querying the V\$LIBRARY Cache Table Monitor the statistics in the V\$LIBRARYCACHE table over a period of time with this query:

```
SELECT SUM(pins) "Executions",
       SUM(reloads) "Cache Misses while Executing"
FROM v$librarycache
```

The output of this query might look like this:

Executions	Cache Misses while Executing
-----	-----
320871	549

Interpreting the V\$LIBRARYCACHE Table Examining the data returned by the sample query leads to these observations:

- The sum of the PINS column indicates that SQL statements, PL/SQL blocks, and object definitions were accessed for execution a total of 320,871 times.
- The sum of the RELOADS column indicates that 549 of those executions resulted in library cache misses causing ORACLE to implicitly reparse a statement or block or reload an object definition because it had aged out of the library cache.
- The ratio of the total RELOADS to total PINS is about 0.17%. This value means that only 0.17% of executions resulted in reparsing.

Total RELOADS should be near 0. If the ratio of RELOADS to PINS is more than 1%, then you should reduce these library cache misses through the means discussed in the next section.

Reducing Library Cache Misses

You can reduce library cache misses by:

- allocating additional memory for the library cache
- writing identical SQL statements whenever possible

Allocating Additional Memory for the Library Cache You may be able to reduce library cache misses on execution calls by allocating additional memory for the library cache. To ensure that shared SQL areas remain in the cache once their SQL statements are parsed, increase the amount of memory available to the library cache until the V\$LIBRARYCACHE.RELOADS value is near 0. To increase the amount of memory available to the library cache, increase the value of the initialization parameter SHARED_POOL_SIZE. The maximum value for this parameter depends on your operating system. This measure will reduce implicit reparsing of SQL statements and PL/SQL blocks on execution.

IUG

To take advantage of additional memory available for shared SQL areas, you may also need to increase the number of cursors permitted for a session. You can increase this limit by increasing the value of the initialization parameter OPEN_CURSORS.

Be careful not to induce paging and swapping by allocating too much memory for the library cache. The benefits of a library cache large enough to avoid cache misses can be partially offset by reading shared SQL areas into memory from disk whenever you need to access them.

Writing Identical SQL Statements You may be able to reduce library cache misses on parse calls by ensuring that SQL statements and PL/SQL blocks share a shared SQL area whenever possible. For two different occurrences of a SQL statement or PL/SQL block to share a shared SQL area, they must be identical according to these criteria:

- The text of the SQL statements or PL/SQL blocks must be identical, character for character, including spaces and case.

For example, these statements cannot use the same shared SQL area

```
SELECT * FROM emp
SELECT *   FROM emp
```

These statements cannot use the same shared SQL area:

```
SELECT * FROM emp
SELECT * FROM Emp
```

- References to schema objects in the SQL statements or PL/SQL blocks must resolve to the same object in the same schema.

For example, if the schemas of the users BOB and ED both contain an EMP table and both users issue the following statement, their statements cannot use the same shared SQL area

```
SELECT * FROM emp
```

If both statements query the same table and qualify the table with the schema, as in the following statement, then they can use the same shared SQL area:

```
SELECT * FROM bob.emp
```

- Bind variables in the SQL statements must match in name and datatype. For example, these statements cannot use the same shared SQL area:

```
SELECT * FROM emp WHERE deptno = :department_no
SELECT * FROM emp WHERE deptno = :d_no
```

- The SQL statements must be optimized using the same optimization approach and, in the case of the cost-based approach, the same optimization goal. For information on optimization approach and goal, see Chapter 5 “Tuning SQL Statements” of the *ORACLE7 Server Application Developer’s Guide*.



Shared SQL areas are most useful for reducing library cache misses for multiple users running the same application. Discuss these criteria with the developers of such applications and agree on strategies to ensure that the SQL statements and PL/SQL blocks of an application can use the same shared SQL areas:

- Use bind variables rather than explicitly specified constants in your statements whenever possible.

For example, the following two statements cannot use the same shared area because they do not match character for character

```
SELECT ename, empno FROM emp WHERE deptno = 10
SELECT ename, empno FROM emp WHERE deptno = 20
```

You can accomplish the goals of these statements by using the following statement that contains a bind variable and binding 10 for one occurrence of the statement and 20 for the other:

```
SELECT ename, empno FROM emp WHERE deptno = :department_no
```

The two occurrences of the statement can use the same shared SQL area.

- Be sure that individual users of the application do not change the optimization approach and goal for their individual sessions.

You can also increase the likelihood that SQL statements issued by different applications can share SQL areas by establishing these policies among the developers of these applications:

- Standardize naming conventions for bind variables and spacing conventions for SQL statements and PL/SQL blocks.
- Use stored procedures whenever possible. Multiple users issuing the same stored procedure automatically use the same shared PL/SQL area. Since stored procedures are stored in a parsed form, they eliminate runtime parsing altogether.

If you have no library cache misses, you may still be able to speed execution calls by setting the value of the initialization parameter `CURSOR_SPACE_FOR_TIME`. This parameter specifies when a shared SQL area can be deallocated from the library cache to make room for a new SQL statement. The default value of this parameter is `FALSE`, meaning that a shared SQL area can be deallocated from the library cache regardless of whether application cursors associated with its SQL statement are open. The value of `TRUE` means that a shared SQL area can only be deallocated when all application cursors associated with its statement are closed.

Depending on the value of `CURSOR_SPACE_FOR_TIME`, ORACLE behaves differently when an application makes an execution call. If the value is `FALSE`, ORACLE must take time to check that a shared SQL area containing the SQL statement is in the library cache. If the value is `TRUE`, ORACLE need not make this check because the shared SQL area can never be deallocated while an application cursor associated with it is open. Setting the value of the parameter to `TRUE` saves ORACLE a small amount of time and may slightly improve the performance of execution calls. This value also prevents the deallocation of private SQL areas until associated application cursors are closed.

Do not set the value of `CURSOR_SPACE_FOR_TIME` to `TRUE` if there are library cache misses on execution calls. Such library cache misses indicate that the shared pool is not large enough to hold the shared SQL areas of all concurrently open cursors. If the value is `TRUE` and there is no space in the shared pool for a new SQL statement, the statement cannot be parsed and ORACLE returns an error saying that there is no more shared memory. If the value is `FALSE` and there is no space for a new statement, ORACLE deallocates an existing shared SQL area. Although deallocating a shared SQL area results in a library cache miss later, it is preferable to an error halting your application because a SQL statement cannot be parsed.

Do not set the value of `CURSOR_SPACE_FOR_TIME` to `TRUE` if the amount of memory available to each user for private SQL areas is scarce. This value also prevents the deallocation of private SQL areas associated with open cursors. If the private SQL areas for all concurrently open cursors fills the user's available memory so that there is no space to allocate a private SQL area for a new SQL statement, the statement cannot be parsed and ORACLE returns an error indicating that there is not enough memory.

**Tuning the Data
Dictionary Cache**

In this section, you will tune the data dictionary cache. These topics are discussed in this section:

- how to monitor the activity of the data dictionary cache
- how to improve the performance of the data dictionary cache

Examining Data
Dictionary Cache Activity

Determine whether misses on the data dictionary cache are affecting the performance of ORACLE. You can examine cache activity by querying the V\$ROWCACHE table as described in the following sections.

Misses on the data dictionary cache are to be expected in some cases. Upon instance startup, the data dictionary cache contains no data, so any SQL statement issued is likely to result in cache misses. As more data is read into the cache, the likelihood of cache misses should decrease. Eventually the database should reach a “steady state” in which the most frequently used dictionary data is in the cache. At this point, very few cache misses should occur. To tune the cache, examine its activity only after your application has been running.

The V\$ROWCACHE Table Statistics reflecting data dictionary activity are kept in the dynamic performance table V\$ROWCACHE. By default, this table is only available to the user SYS and to users granted SELECT ANY TABLE system privilege, such as SYSTEM.

Each row in this table contains statistics for a single type of the data dictionary item. These statistics reflect all data dictionary activity since the most recent instance startup. These columns in the V\$ROWCACHE table reflect the use and effectiveness of the data dictionary cache:

PARAMETER	This column identifies a particular data dictionary item. For each row, the value in this column is the item prefixed by ‘dc_’.
	For example, in the row that contains statistics for file descriptions, this column has the value ‘dc_files’.
GETS	This column shows the total number of requests for information on the corresponding item.
	For example, in the row that contains statistics for file descriptions, this column has the total number of requests for file descriptions data.
GETMISSES	This column shows the number of data requests resulting in cache misses.

Querying the V\$ROWCACHE Table Monitor the statistics in the V\$ROWCACHE table over a period of time while your application is running with this query:

```
SELECT SUM(gets)      "Data Dictionary Gets",
       SUM(getmisses) "Data Dictionary Cache Get Misses"
FROM v$rowcache
```

The output of this query might look like this:

Data Dictionary Gets	Data Dictionary Cache Get Misses
1439044	3120

Interpreting the V\$ROWCACHE Table Examining the data returned by the sample query leads to these observations:

- The sum of the GETS column indicates that there were a total of 1,439,044 requests for dictionary data.
- The sum of the GETMISSES column indicates that 3120 of the requests for dictionary data resulted in cache misses.
- The ratio of the sums of GETMISSES to GETS is about 0.2%.

Reducing Data Dictionary Cache Misses

Examine cache activity by monitoring the sums of the GETS and GETMISSES columns. For frequently accessed dictionary caches, the ratio of total GETMISSES to total GETS should be less than 10% or 15%. If this ratio continues to increase above this threshold while your application is running, you should consider increasing the amount of memory available to the data dictionary cache. To increase the memory available to the cache, increase the value of the initialization parameter SHARED_POOL_SIZE. The maximum value for this parameter varies depending on your operating system.



Tuning the Shared Pool with the Multi-Threaded Server

In the multi-threaded server architecture, ORACLE stores session information in the shared pool rather than in the memory of user processes. Session information includes private SQL areas and sort areas. If you are using the multi-threaded server architecture, you may need to make your shared pool larger to accommodate session information. You can increase the size of the shared pool by increasing the value of the SHARED_POOL_SIZE initialization parameter. This section discusses measuring the size of session information by querying the dynamic performance table V\$SESSTAT.

The V\$SESSTAT Table

ORACLE collects statistics on total memory used by a session and stores them in the dynamic performance table V\$SESSTAT. By default, this table is only available to the user SYS and to users granted SELECT ANY TABLE system privilege, such as SYSTEM. These statistics are useful for measuring session memory use:

session memory The value of this statistic is the amount of memory in bytes allocated to the session.

max session memory The value of this statistic is the maximum amount of memory in bytes ever allocated to the session.

Querying the V\$SESSTAT Table

You can use this query to decide how much larger to make the shared pool if you are using the multi-threaded server. Issue these queries while your application is running:

```
SELECT SUM (value) || ' bytes' "Total memory for all sessions"
  FROM v$sesstat
 WHERE name = 'session memory'
```

```
SELECT SUM (value) || ' bytes' "Total max mem for all sessions"
  FROM v$sesstat
 WHERE name = 'max session memory'
```

The results of these queries might look like this:

Total memory for all sessions

157125 bytes

Total max mem for all sessions

417381 bytes

Interpreting the V\$SESSTAT Table

The result of the first query indicates that the memory currently allocated to all sessions is 15,715 bytes. This value is the total memory whose location depends on how the sessions are connected to ORACLE. If the sessions are connected with dedicated servers processes, this memory is part of the memories of the user processes. If the sessions are connected with shared server processes, this memory is part of the shared pool. The result of the second query indicates the sum of the maximum sizes of the memories for all sessions is 417,381 bytes. The second result is greater than the first because some sessions have deallocated memory since allocating their maximum amounts.

You can use the result of either of these queries to determine how much larger to make the shared pool if you use the multi-threaded server. The first value is likely to be a better estimate than the second unless nearly all sessions are likely to reach their maximum allocations at the same time.

Tuning the Buffer Cache

In this section, you will learn how to tune the buffer cache. The following issues are covered in this section:

- how to monitor buffer cache performance
- how to improve buffer cache performance

Examining Buffer Cache Activity

ORACLE collects statistics that reflect data access and stores them in the dynamic performance table V\$SYSSTAT. By default, this table is only available to the user SYS and to users granted SELECT ANY TABLE system privilege, such as SYSTEM. These statistics are useful for tuning the buffer cache:

<i>db block gets,</i> <i>consistent gets</i>	The sum of the values of these statistics is the total number of requests for data. This value includes requests satisfied by access to buffers in memory.
<i>physical reads</i>	The value of this statistic is the total number of requests for data resulting in access to data files on disk.

Monitor these statistics over a period of time while your application is running with this query:

```
SELECT name, value
      FROM v$sysstat
     WHERE name IN ('db block gets', 'consistent gets', 'physical reads')
```

The output of this query might look like this:

NAME	VALUE
db block gets	85792
consistent gets	278888
physical reads	23182

Calculate the hit ratio for the buffer cache with this formula:

Hit Ratio = 1 - (*physical reads* / (*db block gets* + *consistent gets*))

Based on the statistics obtained by the example query, the buffer cache hit ratio is 94%.

Reducing Buffer Cache Misses

If your hit ratio is low, say less than 60% or 70%, then you may want to increase the number of buffers in the cache to improve performance. To make the buffer cache larger, increase the value of the initialization parameter `DB_BLOCK_BUFFERS`. The maximum value for this parameter is 65535.

ORACLE can collect statistics that estimate the performance gain that would result from increasing the size of your buffer cache. With these statistics, you can estimate how many buffers to add to your cache.

The X\$KCBRBH Table

The virtual table `SYS.X$KCBRBH` contains statistics that estimate the performance of a larger cache. Each row in the table reflects the relative performance value of adding a buffer to the cache. This table can only be accessed by the user `SYS`. The following are the columns of the `X$KCBRBH` table:

INDX	The value of this column is one less than the number of buffers that would potentially be added to the cache.
COUNT	The value of this column is the number of additional cache hits that would be obtained by adding additional buffer number <code>INDX+1</code> to the cache.

For example, in the first row of the table, the `INDX` value is 0 and the `COUNT` value is the number of cache hits to be gained by adding the first additional buffer to the cache. In the second row, the `INDX` value is 1 and the `COUNT` value is the number of cache hits for the second additional buffer.

Enabling the X\$KCBRBH Table

The collection of statistics in the `X$KCBRBH` table is controlled by the initialization parameter `DB_BLOCK_LRU_EXTENDED_STATISTICS`. The value of this parameter determines the number of rows in the `X$KCBRBH` table. The default value of this parameter is 0, which means the default behavior is to not collect statistics.

To enable the collection of statistics in the `X$KCBRBH` table, set the value of `DB_BLOCK_LRU_EXTENDED_STATISTICS`. For example, if you set the value of the parameter to 100, ORACLE will collect 100 rows of statistics, each row reflecting the addition of one buffer, up to 100 extra buffers.

Collecting these statistics incurs some performance overhead. This overhead is proportional to the number of rows in the table. To avoid this overhead, collect statistics only when you are tuning the buffer cache and disable the collection of statistics when you are finished tuning.

Querying the X\$KCBRBH Table

From the information in the X\$KCBRBH table, you can predict the potential gains of increasing the cache size. For example, to determine how many more cache hits would occur if you added 20 buffers to the cache, query the X\$KCBRBH table with the SQL statement:

```
SELECT SUM(count) ach
FROM sys.x$kcbrbh
WHERE indx < 20
```

You can also determine how these additional cache hits would affect the hit ratio. Use the following formula to calculate the hit ratio based on the values of the statistics *db block gets*, *consistent gets*, and *physical reads* and the number of additional cache hits (ACH) returned by the query:

$$Hit\ Ratio = 1 - (physical\ reads - ACH / (db\ block\ gets + consistent\ gets))$$

Grouping Rows in the X\$KCBRBH Table

Another way to examine the X\$KCBRBH table is to group the additional buffers in large intervals. You can query the table with a SQL statement similar to this one:

```
SELECT 250*TRUNC(indx/250)+1|| ' to ' ||250*(TRUNC(indx/250)+1)
"Interval", SUM(count) "Buffer Cache Hits"
FROM sys.x$kcbrbh
GROUP BY TRUNC(indx/250)
```

The result of this query might look like this:

Interval	Buffer Cache Hits
-----	-----
1 to 250	16080
251 to 500	10950
501 to 750	710
751 to 1000	23140

where:

- INTERVAL is the interval of additional buffers to be added to the cache.
- BUFFER CACHE HITS is the number of additional cache hits to be gained by adding the buffers in the INTERVAL column.

Examining the query output leads to these observations:

- If 250 buffers were added to the cache, 16,080 cache hits would be gained.
- If 250 more buffers were added for a total of 500 additional buffers, 10,950 cache hits would be gained in addition to the 16,080 cache hits from the first 250 buffers. This means that adding 500 buffers would yield a total of 27,030 additional cache hits.
- If 250 more buffers were added for a total of 750 additional buffers, 710 cache hits would be gained, yielding a total of 27,740 additional cache hits.
- If 250 buffers were added to the cache for a total of 1000 additional buffers, 23,140 cache hits would be gained, yielding a total of 50,880 additional cache hits.

Based on these observations, you should decide how many buffers to add to the cache. In this case, you may make these decisions:

- It is wise to add 250 or 500 buffers, provided memory resources are available. Both of these increments offer significant performance gains.
- It is unwise to add 750 buffers. Nearly the entire performance gain made by such an increase can be made by adding 500 buffers instead. Also, the memory allocated to the additional 250 buffers may be better used by some other ORACLE memory structure.
- It is wise to add 1000 buffers, provided memory resources are available. The performance gain from adding 1000 buffers to the cache is significantly greater than the gains from adding 250, 500, or 750 buffers.

Removing Unnecessary Buffers

If your hit ratio is high, your cache is probably large enough to hold your most frequently accessed data. In this case, you may be able to reduce the cache size and still maintain good performance. To make the buffer cache smaller, reduce the value of the initialization parameter `DB_BLOCK_BUFFERS`. The minimum value for this parameter is 4. You can apply any leftover memory to other ORACLE memory structures.

ORACLE can collect statistics to predict buffer cache performance based on a smaller cache size. Examining these statistics can help you determine how small you can afford to make your buffer cache without adversely affecting performance.

The X\$KCBCBH Table

The virtual table `SYS.X$KCBCBH` contains the statistics that estimate the performance of a smaller cache. The `X$KCBCBH` table is similar in structure to the `X$KCBRBH` table. This table can only be accessed by the user `SYS`. The following are the columns of the `X$KCBCBH` table:

<code>INDX</code>	The value of this column is the potential number of buffers in the cache.
<code>COUNT</code>	The value of this column is the number of cache hits attributable to buffer number <code>INDX</code> .

The number of rows in this table is equal to the number of buffers in your buffer cache. Each row in the table reflects the number of cache attributed to a single buffer. For example, in the second row, the `INDX` value is 1 and the `COUNT` value is the number of cache hits for the second buffer. In the third row, the `INDX` value is 2 and the `COUNT` value is the number of cache hits for the third buffer.

The first row of the table contains special information. The `INDX` value is 0 and the `COUNT` value is the total number of blocks moved into the first buffer in the cache.

Enabling the X\$KCBCBH Table

The collection of statistics in the `X$KCBCBH` table is controlled by the initialization parameter `DB_BLOCK_LRU_STATISTICS`. The value of this parameter determines whether ORACLE collects the statistics. The default value for this parameter is `FALSE`, which means that the default behavior is to not collect statistics.

To enable the collection of statistics in the `X$KCBCBH` table, set the value of `DB_BLOCK_LRU_STATISTICS` to `TRUE`.

Collecting these statistics incurs some performance overhead. To avoid this overhead, collect statistics only when you are tuning the buffer cache and disable the collection of statistics when you are finished tuning.

Querying the X\$KCBCBH Table

From the information in the X\$KCBCBH table, you can predict the number of additional cache misses that would occur if the number of buffers in the cache were reduced. If your buffer cache currently contains 100 buffers, you may want to know how many more cache misses would occur if it had only 90. To determine the number of additional cache misses, query the X\$KCBCBH table with the SQL statement

```
SELECT SUM(count) acm
FROM sys.x$kcbbh
WHERE indx >= 90
```

You can also determine the hit ratio based on this cache size. Use the following formula to calculate the hit ratio based on the values of the statistics *db block gets*, *consistent gets*, and *physical reads* and the number of additional cache misses (ACM) returned by the query:

Hit Ratio = 1 - (*physical reads* + ACM / (*db block gets* + *consistent gets*))

Grouping Rows in the X\$KCBCBH Table

Another way to examine the X\$KCBCBH table is to group the buffers in intervals. For example, if your cache contains 100 buffers, you may want to divide the cache into four 25-buffer intervals. You can query the table with a SQL statement similar to this one:

```
SELECT 25*TRUNC(indx/25)+1|| ' to ' ||25*(TRUNC(indx/25)+1) "Interval",
SUM(count) "Buffer Cache Hits"
FROM sys.x$kcbbh
WHERE indx > 0
GROUP BY TRUNC(indx/25)
```

Note that the WHERE clause prevents the query from collecting statistics from the first row of the table. The result of this query might look like this:

Interval	Buffer Cache Hits
-----	-----
1 to 25	1900
26 to 50	1100
51 to 75	1360
76 to 100	230

where:

- INTERVAL is the interval of buffers in the cache.
- BUFFER CACHE HITS is the number of cache hits attributable to the buffers in the INTERVAL column.

Examining the query output leads to these observations:

- The last 25 buffers in the cache (buffers 76 to 100) contribute 230 cache hits. If the cache were reduced in size by 25 buffers, 230 cache hits would be lost.
- The third 25-buffer interval (buffers 51 to 75) contributes 1,360 cache hits. If these buffers were removed from the cache, 1,360 cache hits would be lost in addition to the 230 cache hits lost for buffers 76 to 100. Removing 50 buffers would result in losing a total of 1,590 cache hits.
- The second 25-buffer interval (buffers 26 to 50) contributes 1,100 cache hits. Removing 75 buffers from the cache would result in losing a total of 2,690 cache hits.
- The first 25 buffers in the cache (buffers 1 to 25) contribute 1,900 cache hits.

Based on these observations, you should decide whether to reduce the size of the cache. In this case, you may make these decisions:

- If memory is scarce, it may be wise to remove 25 buffers from the cache. The buffers 76 to 100 contribute relatively few cache hits compared to the total cache hits contributed by the entire cache. Removing 25 buffers will not significantly reduce cache performance, and the leftover memory may be better used by other ORACLE memory structures.
- It is unwise to remove more than 25 buffers from the cache. For example, removing 50 buffers would reduce cache performance significantly. The cache hits contributed by these buffers is a significant portion of the total cache hits.

Reallocating Memory

After resizing your ORACLE memory structures, re-evaluate the performance of the library cache, the data dictionary cache, and the buffer cache. If you have reduced the memory consumption of any one of these structures, you may want to allocate more memory to another structure. For example, if you have reduced the size of your buffer cache, you may now want to take advantage of the additional available memory by using it for the library cache.

Tune your operating system again. Resizing ORACLE memory structures may have changed ORACLE memory requirements. In particular, be sure paging and swapping is not excessive. For example, if the size of the data dictionary cache or the buffer cache has increased, the SGA maybe too large to fit into main memory. In this case, the SGA could be paged or swapped.

In the process of reallocating memory, you may determine that the optimum size of ORACLE memory structures requires more memory than your operating system can provide. In this case, you may improve performance even further by adding more memory to your computer.

TUNING I/O

This chapter presents Step Two of the tuning process: tuning I/O. This chapter teaches you how to avoid I/O bottlenecks that could prevent ORACLE from performing at its maximum potential. In this chapter you learn how to:

- reduce disk contention
- allocate space in data blocks
- avoid dynamic space management

The Importance of Tuning I/O

The performance of many software applications is inherently limited by disk I/O. Often, CPU activity must be suspended while I/O activity completes. Such an application is said to be “I/O bound.” ORACLE is designed so that performance need not be limited by I/O.

Tuning I/O can help performance if a disk containing database files is operating at its capacity. However, tuning I/O cannot help performance in “CPU bound” cases, or cases in which your computer’s CPUs are operating at their capacity.

It is important to tune I/O after following the memory allocation recommendations presented in Chapter 21 “Tuning Memory Allocation” of this Guide. Chapter 21 shows you how to allocate memory in order to reduce I/O to a minimum. After reaching this minimum, follow the instructions in this chapter in order to perform the necessary I/O as efficiently as possible.

Reducing Disk Contention

In this section you will learn how to reduce disk contention. The following issues are discussed in this section:

- what disk contention is
- how to monitor disk activity
- how to reduce disk activity

What Is Disk Contention?

Disk contention occurs when multiple processes try to access the same disk simultaneously. Most disks have limits on both the number of accesses and the amount of data they can transfer per second. When these limits are reached, processes may have to wait to access the disk.

Monitoring Disk Activity

Disk activity is reflected by:

- ORACLE file I/O statistics
- operating system statistics

ORACLE compiles ORACLE file I/O statistics that reflect disk access to database files. Your operating system may also keep statistics for disk access to all files.

Monitoring ORACLE Disk Activity

Examine disk access to database files through the dynamic performance table V\$FILESTAT. By default, this table is only available to the user SYS and to users granted SELECT ANY TABLE system privilege, such as SYSTEM. These column values reflect the number of disk accesses for each datafile:

PHYRDS The value of this column is the number of reads from each database file.

PHYWRTS This value of this column is the number of writes to each database file.

Monitor these values over some period of time while your application is running with this query:

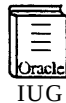
```
SELECT name, phylds, phywrts
       FROM v$datafile df, v$filestat fs
       WHERE df.file# = fs.file#
```

This query also retrieves the name of each data file from the dynamic performance table V\$DATAFILE. The output of this query might look like this:

NAME	PHYRDS	PHYWRTS
DISK\$DEV20:[ORACLE.DB_ORA70}ORA_SYSTEM.DBS	7679	2735
DISK\$DEV20:[ORACLE.DB_ORA70}ORA_SYSTEM1.DBS	32	546

The total I/O for a single disk is the sum of PHYRDS and PHYWRTS for all the database files managed by the ORACLE instance on that disk. Determine this value for each of your disks. Also determine the rate at which I/O occurs for each disk by dividing the total I/O by the interval of time over which the statistics were collected.

Monitoring Operating System Disk Activity



Disks holding data files and redo log files may also hold files that are not related to ORACLE. Access to such files can only be monitored through operating system facilities rather than through the V\$FILESTAT table. Such facilities maybe documented in either the ORACLE installation or user's guide for your operating system or your operating system documentation.

Use your operating system facilities to examine the total I/O to your disks. Try to reduce any heavy access to disks that contain database files.

Distributing I/O

Consider the statistics in the V\$FILESTAT table and your operating system facilities. Consult your hardware documentation to determine the limits on the capacity of your disks. Any disks operating at or near full capacity are potential sites for disk contention. For example, 40 or more I/Os per second is excessive for most disks on VMS or UNIX operating systems.

To reduce the activity on an overloaded disk, move one or more of its heavily accessed files to a less active disk. Apply this principle to each of your disks until they all have roughly the same amount of I/O. This is referred to as *distributing I/O*.

This section discusses guidelines for distributing I/O:

- Separate data files and redo log files on different disks.
- Separate, or “stripe,” table data on different disks.
- Separate tables and indexes on different disks.
- Reduce disk I/O not related to ORACLE.

Separating Data Files and Redo Log Files

ORACLE processes constantly access data files and redo log files. If these files are on common disks, there is potential for disk contention.

Place each data file on a separate disk. Multiple processes can then access different files concurrently without disk contention.

Place each set of redo log files on a separate disk with no other activity. Redo log files are written by the Log Writer process (LGWR) when a transaction is committed. Information in a redo log file is written sequentially. This sequential writing can take place much faster if there is no concurrent activity on the same disk.

Dedicating a separate disk to redo log files usually ensures that LGWR runs smoothly with no further tuning attention. Performance bottlenecks related to LGWR are rare. For information on tuning LGWR, see the section “Reducing Contention for Redo Log Buffer Latches” on page 23-9.

Note that *mirroring* redo log files, or maintaining multiple copies of each redo log file on separate disks, does not slow LGWR considerably. LGWR writes to each disk in parallel and waits until each part of the parallel write is complete. Since the time required for your operating system to perform a single-disk write may vary, increasing the number of copies increases the likelihood that one of the single-disk writes in the parallel write will take longer than average. A parallel write will not take longer than the longest possible single-disk write. There may also be some overhead associated with parallel writes on your operating system.



Dedicating separate disks and mirroring redo log files are important safety precautions. Dedicating separate disks to data files and redo log files ensures that both the data files and the redo log files cannot be lost in a single disk failure. Mirroring redo log files ensures that a single redo log file cannot be lost in a single disk failure.

“Striping” Table Data

“Striping” is the practice of dividing a large table’s data into small portions and storing these portions in separate data files on separate disks. This permits multiple processes to access different portions of the table concurrently without disk contention. “Striping” is particularly helpful in optimizing random access to tables with many rows.

To create a “striped” table:

1. Create a tablespace with the CREATE TABLESPACE command. Specify the data files in the DATAFILE clause. Each of the files should be on a different disk.

```
CREATE TABLESPACE stripedtablespace
  DATAFILE 'file_on_disk_1' SIZE 500K,
           'file_on_disk_2' SIZE 500K,
           'file_on_disk_3' SIZE 500K,
           'file_on_disk_4' SIZE 500K,
           'file_on_disk_5' SIZE 500K
```

2. Then create the table with the CREATE TABLE command. Specify the newly created tablespace in the TABLESPACE clause.

Also specify the size of the table extents in the STORAGE clause. Store each extent in a separate data file. The table extents should be slightly smaller than the data files in the tablespace to allow for overhead. For example:

```
CREATE TABLE stripedtab
  ( col_1 NUMBER(2),
    col_2 VARCHAR2(10) )
  TABLESPACE stripedtablespace
  STORAGE ( INITIAL 495K NEXT 495K
           MINEXTENTS 5 PCTINCREASE 0 )
```

These steps result in the creation of table STRIPEDTAB. STRIPEDTAB has 5 initial extents, each of size 495 kilobytes. Each extent takes up one of the data files named in the DATAFILE clause of the CREATE TABLESPACE statement. These files are all on separate disks. These 5 extents are all allocated immediately, since MINEXTENTS is 5. For more information on MINEXTENTS and the other storage parameters, see Chapter 8 “Managing Schema Objects” of this Guide.

Separating Tables and Indexes

Place frequently accessed database structures in separate data files on separate disks. To do this, you must know which of your database structures are used often. For example, separate an often used table from its index. This separation distributes the I/O to the table and index across separate disks.

Follow these steps to separate a table and its index:

1. Create a tablespace with the CREATE TABLESPACE command. Specify the data file in the DATAFILE clause:

```
CREATE TABLESPACE tabspace_1
      DATAFILE 'file_on_disk_1'
```

2. Create the table with the CREATE TABLE command. Specify the tablespace in the TABLESPACE clause:

```
CREATE TABLE tab_1
      ( col_1  NUMBER(2),
        col_2  VARCHAR2(10) )
TABLESPACE tabspace_1
```

3. Create another tablespace. Specify a data file on another disk:

```
CREATE TABLESPACE tabspace_2
      DATAFILE 'file_on_disk_2'
```

4. Create the index. Specify the new tablespace:

```
CREATE INDEX ind_1 ON tab_1 (col_1)
      TABLESPACE tabspace_2
```

These steps result in the creation of table TAB_1 in the file FILE_ON_DISK_1 and the creation of index IND_1 in the file FILE_ON_DISK_2.

Eliminating Other Disk I/O

If possible, eliminate I/O not related to ORACLE on disks that contain database files. This measure is especially helpful in optimizing access to redo log files. Not only does this reduce disk contention, it also allows you to monitor all activity on such disks through the dynamic performance table V\$FILESTAT.

Allocating Space in Data Blocks

Table data in the database is stored in data blocks. In this section, you will learn how to allocate space within data blocks for best performance. The following issues are discussed in this section:

- how to control data storage
- how to store data most efficiently based on your application

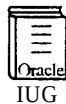
Migrated and Chained Rows

If an UPDATE statement increases the amount of data in a row so that the row no longer fits in its data block, ORACLE tries to find another block with enough free space to hold the entire row. If such a block is available, ORACLE moves the entire row to the new block. This is called *migrating* a row. If the row is too large to fit into any available block, ORACLE splits the row into multiple pieces and stores each piece in a separate block. This is called *chaining* a row. Rows can also be chained when they are inserted.

Dynamic space management, especially migration and chaining, is detrimental to performance:

- UPDATE statements that cause migration and chaining perform poorly.
- Queries that select migrated or chained rows must perform more I/O.

You can identify migrated and chained rows in a table or cluster by using the ANALYZE command with the LIST CHAINED ROWS option. This command collects information about each migrated or chained row and places this information into a specified output table. The definition of a sample output table named CHAINED_ROWS appears in a SQL script available on your distribution media. The common name of this script is UTLCHAIN.SQL, although its exact name and location may vary depending on your operating system. Your output table must have the same column names, datatypes, and sizes as the CHAINED_ROWS table.



To reduce migrated and chained rows in an existing table, follow these steps:

1. Use the ANALYZE command to collect information about migrated and chained rows. For example:

```
ANALYZE
    TABLE order_hist LIST CHAINED ROWS
```

2. Query the output table:

```
SELECT *
    FROM chained_rows
    WHERE table_name = 'ORDER_HIST'
```

OWNER_NAME	TABLE_NAME	CLUSTER_NAME	HEAD_ROWID	TIMESTAMP
SCOTT	ORDER_HIST		0000186A.0003.0001	04-AUG-92
SCOTT	ORDER_HIST		0000186A.0002.0001	04-AUG-92
SCOTT	ORDER_HIST		0000186A.0001.0001	04-AUG-92

3. If the output table shows that you have many migrated or chained rows, follow these steps to eliminate migrated rows:

- a. Create an intermediate table with the same columns as the existing table to hold the migrated and chained rows:

```
CREATE TABLE int_order_hist
AS SELECT *
    FROM order_hist
    WHERE ROWID IN
        (SELECT head_rowid
         FROM chained_rows
         WHERE table_name = 'ORDER_HIST')
```

- b. Delete the migrated and chained rows from the existing table:

```
DELETE FROM order_hist
    WHERE ROWID IN
        (SELECT head_rowid
         FROM chained_rows
         WHERE table_name = 'ORDER_HIST')
```

- c. Insert the rows of the intermediate table into the existing table:

```
INSERT INTO order_hist
    SELECT *
    FROM int_order_hist
```

- d. Drop the intermediate table:

```
DROP TABLE int_order_history
```

4. Delete the information collected in step 1 from the output table:

```
DELETE FROM chained_rows
WHERE table_name = 'ORDER_HIST'
```
5. Use the ANALYZE command again and query the output table.
6. Any rows that appear in the output table are chained rows. You can eliminate chained rows by increasing your data block size. It may not be possible to avoid chaining in all situations. If your table has a LONG column or long CHAR or VARCHAR2 columns, chaining is often unavoidable.

Avoiding Dynamic Space Management

When an object such as a table or rollback segment is created, space is allocated in the database for the data. This space is called a segment. If subsequent database operations cause the data to grow and exceed the space allocated, ORACLE extends the segment. Dynamic extension can reduce performance. This section discusses:

- how to detect dynamic extension
- how to allocate enough space for your data to avoid dynamic extension
- how to avoid dynamic space management in rollback segments

Detecting Dynamic Extension

Dynamic extension causes ORACLE to execute SQL statements in addition to those SQL statements issued by user processes. These SQL statements are called *recursive calls*, because ORACLE issues these statements itself. Recursive calls are also generated by these activities:

- misses on the data dictionary cache
- firing of database triggers
- execution of Data Definition Language statements
- execution of SQL statements within stored procedures, functions, packages, and anonymous PL/SQL blocks
- enforcement of referential integrity constraints

Examine the *recursive calls* statistic through the dynamic performance table V\$SYSSTAT. By default, this table is only available to the user SYS and to users granted SELECT ANY TABLE system privilege, such as SYSTEM. Monitor this statistic over some period of time while your application is running with this query:

```
SELECT name, value
       FROM v$sysstat
       WHERE name = 'recursive calls'
```

The output of this query might look like this:

NAME	VALUE
recursive calls	626681

If ORACLE continues to make an excess of recursive calls while your application is running, determine whether these recursive calls are due to one of the activities that generate recursive calls other than dynamic extension. If you determine that these recursive calls are caused by dynamic extension, you should try to reduce this extension by allocating larger extents.

Allocating Extents

Follow these steps to avoid dynamic extension:

1. Determine the maximum size of your object. For formulas to predict how much space to allow for a table, see Chapter 8 “Managing Schema Objects” of this Guide.
2. Choose storage parameter values so that ORACLE allocates extents large enough to accommodate all of your data when you create the object.

Larger extents tend to benefit performance for these reasons:

- Since blocks in a single extent are contiguous, one large extent is more contiguous than multiple small extents. ORACLE can read one large extent from disk with fewer multi-block read than would be required to read many small extents.
- segments with larger extents are less likely to be extended.

However, since large extents require more contiguous blocks, ORACLE may have difficulty finding enough contiguous space to store them. To determine whether to allocate few large extents or many small extents, consider the benefits and drawbacks of each in light of your plans for the growth and use of your tables.

Avoiding Dynamic Space Management in Rollback Segments

The size of rollback segments can also affect performance. Rollback segment size is determined by the rollback segment's storage parameter values. Your rollback segments must be large enough to hold the rollback entries for your transactions. As with other objects, you should avoid dynamic space management in rollback segments.

Use the SET TRANSACTION command to assign transactions to rollback segments of the appropriate size based on the recommendations in the following sections. If you do not explicitly assign a rollback segment to a transaction, ORACLE randomly chooses a rollback segment for it. If you are running multiple concurrent copies of the same application, be careful not to assign the transactions for all copies to the same rollback segment. This leads to contention for that rollback segment.

Also monitor the shrinking, or dynamic deallocation, of rollback segments based on the OPTIMAL storage parameter. For information on choosing values for this parameter and monitoring rollback segment shrinking, and adjusting OPTIMAL accordingly, see Chapter 9 "Managing Rollback Segments" of this Guide.

Example This statement assigns the current transaction to the rollback segment OLTP_13:

```
SET TRANSACTION USE ROLLBACK SEGMENT oltp_13
```

For Long Queries

Assign large rollback segments to transactions that modify data that is concurrently selected by long queries. Such queries may require access to rollback segments to reconstruct a read-consistent version of the modified data. These rollback segments must be large enough to hold all the rollback entries for the data while the query is running.

For Long Transactions

Assign large rollback segments to long transactions that modify large amounts of data. A large rollback segment can improve the performance of such a transaction. Such transactions generate large rollback entries. If a rollback entry does not fit into a rollback segment, ORACLE extends the segment. Dynamic extension reduces performance and should be avoided whenever possible.

For OLTP Transactions

Some applications perform *online transaction processing*, or OLTP. OLTP applications are characterized by frequent concurrent transactions that each modify small amounts of data. Assign small rollback segments to OLTP transactions provided that their data is not concurrently queried. Small rollback segments are more likely to remain stored in the buffer cache where they can be accessed quickly. A typical OLTP rollback segment might have 2 extents, each on the order of 10 kilobytes in size. To best avoid contention, create many rollback segments and assign each transaction to its own rollback segment.

TUNING CONTENTION

This chapter presents Step Three of the tuning process: tuning contention. Contention occurs when multiple processes try to access the same resource simultaneously. Contention causes processes to wait for access.

ORACLE provides you with methods of handling contention. In this chapter, you will learn how to:

- detect contention that may affect performance
- reduce contention

The ORACLE resources discussed in this chapter include:

- rollback segments
- processes of the multi-threaded server
- redo log buffer latches

Reducing Contention for Rollback Segments

In this section, you will learn how to reduce contention for rollback segments. The following issues are discussed:

- how to identify contention for rollback segments
- how to create rollback segments
- how to assign rollback segments to particular transactions

Identifying Rollback Segment Contention

Contention for rollback segments is reflected by contention for buffers that contain rollback segment blocks. You can determine whether contention for rollback segments is reducing performance by using the dynamic performance table V\$WAITSTAT.

The V\$WAITSTAT contains statistics that reflect block contention. By default, this table is only available to the user SYS and to other users who have SELECT ANY TABLE system privilege, such as SYSTEM. These statistics reflect contention for different classes of block:

<i>system undo header</i>	The value of this statistic is the number of waits for buffers containing header blocks of the SYSTEM rollback segment.
<i>system undo block</i>	The value of this statistic is the number of waits for buffers containing blocks other than header blocks of the SYSTEM rollback segment.
<i>undo header</i>	The value of this statistic is the number of waits for buffers containing header blocks of rollback segments other than the SYSTEM rollback segment.
<i>undo block</i>	The value of this statistic is the number of waits for buffers containing blocks other than header blocks of rollback segments other than the SYSTEM rollback segment.

Monitor these statistics over a period of time while your application is running with this query:

```
SELECT class, count
      FROM v$waitstat
      WHERE class IN ('system undo header', 'system undo block',
                     'undo header', 'undo block')
```

The result of this query might look like this:

CLASS	COUNT
-----	-----
system undo header	2089
system undo block	633
undo header	1235
undo block	942

Compare the number of waits for each class of block with the total number of requests for data over the same period of time. You can monitor the total number of requests for data over a period of time with this query:

```
SELECT SUM (value)
      FROM v$sysstat
      WHERE name IN ('db block gets', 'consistent gets')
```

The output of this query might look like this:

SUM(VALUE)

929530

If the number of waits for any class is greater than 1% of the total number of requests, you should consider creating more rollback segments to reduce contention.

Creating Rollback Segments

To reduce contention for buffers containing rollback segment blocks, create more rollback segments. Table 23-1 shows some general guidelines for choosing how many rollback segments to allocate based on the number of concurrent transactions on your database. These guidelines are appropriate for most application mixes.

TABLE 23-1
Choosing a Number of
Rollback Segments

Number of Concurrent Transactions (<i>n</i>)	Recommended Number of Rollback Segments
$n < 16$	4
$16 \leq n < 32$	8
$32 \leq n$	$n/4$, but no more than 50

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

In this section, you will learn how to reduce contention for these processes used by the ORACLE's multi-threaded server architecture:

- dispatcher processes
- shared server processes

Reducing Contention for Dispatcher Processes

This section discusses these issues:

- how to identify contention for dispatcher processes
- how to add dispatcher processes

Identifying Contention for Dispatcher Processes

Contention for dispatcher processes can be reflected by either of these symptoms:

- high busy rates for existing dispatcher processes
- steady increase in waiting time for responses in the response queues of existing dispatcher processes

Examining Busy Rates for Dispatcher Processes Statistics reflecting the activity of dispatcher processes are kept in the dynamic performance table V\$DISPATCHER. By default, this table is only available to the user SYS and to other users who have SELECT ANY TABLE system privilege, such as SYSTEM. These columns reflect busy rates for dispatcher processes:

IDLE	This column shows the idle time for the dispatcher process in hundredths of seconds.

BUSY	This column shows the busy time for the dispatcher process in hundredths of seconds.

Monitor these statistics over a period of time while your application is running with this query:

```
SELECT network                                "Protocol",
       SUM(busy) / ( SUM(busy) + SUM(idle) ) "Total Busy Rate"
FROM   v$dispatcher
GROUP BY network
```

This query returns the total busy rate for the dispatcher processes of each protocol or the percentage of time the dispatcher processes of each protocol are busy. The result of this query might look like this:

Protocol	Total Busy Rate
decnet	.004589828
tcp	.029111042

From this result, you can make these observations:

- DECnet dispatcher processes are busy nearly .5% of the time.
- TCP dispatcher processes are busy nearly 3% of the time.

If the dispatcher processes for a specific protocol are busy more than 50% of the time, then you maybe able to improve performance for users connected to ORACLE using that protocol by adding dispatcher processes.

Examining Wait Times for Dispatcher Process Response Queues

Statistics reflecting the response queue activity for dispatcher processes are kept in the dynamic performance table V\$QUEUE. By default, this table is only available to the user SYS and to other users who have SELECT ANY TABLE system privilege, such as SYSTEM. These columns wait times for responses in the queue:

WAIT This column shows the total waiting time in hundredths of seconds for all responses that have ever been in the queue.

TOTALQ This column shows the total number of responses that have ever been in the queue.

Monitor these statistics from time to time while your application is running with this query:

```
SELECT network      "Protocol",
       DECODE( SUM(totalq), 0, 'No Responses',
              SUM(wait)/SUM(totalq) || ' hundredths of seconds')
       "Average Wait Time per Response"
FROM v$queue q, v$dispatcher d
WHERE q.type = 'DISPATCHER'
      AND q.paddr = d.paddr
GROUP BY network
```

This query returns the average time in hundredths of seconds that a response waits in each response queue for a dispatcher process to route it to a user process. This query uses the V\$DISPATCHER table to group the rows of the V\$QUEUE table by network protocol. The query also uses the DECODE syntax to recognize those protocols for which there have been no responses in the queue. The result of this query might look like this:

Protocol	Average Wait Time per Response
-----	-----
decnet	.1739130 hundredths of seconds
tcp	No Responses

From this result, you can tell that a response in the queue for DECNET dispatcher processes waits an average of .17 hundredths of a second and that there have been no responses in the queue for TCP dispatcher processes.

If the average wait time for a specific network protocol continues to increase steadily as your application runs, you may be able to improve performance of those user processes connected to ORACLE using that protocol by adding dispatcher processes.

Adding Dispatcher Processes

You can add more dispatcher processes while ORACLE is running through one of these means:

- the SQL*DBA Configure Multi-Threaded Dispatchers Dialogue Box
- the MTS_DISPATCHERS parameter of the ALTER SYSTEM command

For more information on adding dispatcher processes, see Chapter 4 “Managing ORACLE Processes” of this Guide.



The total number of dispatcher processes across all protocols is limited by the value of the initialization parameter MTS_MAX_DISPATCHERS. You may need to increase this value before adding dispatcher processes. The default value of this parameter is 5 and the maximum value varies depending on your operating system.

Reducing Contention for Shared Server Processes

This section discusses these issues:

- how to identify contention for shared server processes
- how to increase the maximum number of shared server processes

Identifying Contention for Shared Server Processes

Contention for shared server processes can be reflected by steady increase in waiting time for requests in the request queue. Statistics reflecting the request queue activity for shared server processes are kept in the dynamic performance table V\$QUEUE. By default, this table is only available to the user SYS and to other users who have SELECT ANY TABLE system privilege, such as SYSTEM. These columns show wait times for requests in the queue:

WAIT	This column shows the total waiting time in hundredths of seconds for all requests that have ever been in the queue.

TOTALQ	This column shows the total number of requests that have ever been in the queue.

Monitor these statistics from time to time while your application is running with this query:

```
SELECT DECODE( totalq, 0, 'No Requests',
               wait/totalq || ' hundredths of seconds')
       "Average Wait Time Per Requests"
FROM v$queue
WHERE type = 'COMMON'
```

This query returns the total wait time for all requests and total number of requests for the request queue. The result of this query might look like this:

```
Average Wait Time per Request
-----
.090909 hundredths of seconds
```

From the result, you can tell that a request waits an average of .09 hundredths of a second in the queue before it is processed.

You can also determine how many shared server processes are currently running by issuing this query:

```
SELECT COUNT(*) "Shared Server Processes"
FROM v$shared_servers
WHERE status != 'QUIT'
```

The result of this query might look like this:

```
Shared Server Processes
-----
10
```

Adding Shared Server Processes



Since ORACLE automatically adds shared server processes if the load on existing shared server processes increases drastically, you are unlikely to improve performance simply by explicitly adding more shared server processes. However, if the number of shared server processes has reached the limit established by the initialization parameter `MTS_MAX_SERVERS` and the average wait time in the requests queue is still increasing, you maybe able to improve performance by increasing the `MTS_MAX_SERVERS` value. The default value of this parameter is 20 and the maximum value varies depending on your operating system. You can then either allow ORACLE to automatically add shared server processes or explicitly add shared processes through one of these means:

- the `MTS_SERVERS` initialization parameter
- the SQL*DBA Configure Multi-Threaded Servers Dialog Box
- the `MTS_SERVERS` parameter of the `ALTER SYSTEM` command

For more information on adding shared server processes, see Chapter 4 “Managing ORACLE Processes” of this Guide.

Reducing Contention for Redo Log Buffer Latches

Contention for redo log buffer access rarely inhibits database performance. However, ORACLE provides you with methods to monitor and reduce any latch contention that does occur. This section explains:

- how to detect and reduce contention for space in the redo log buffer
- how to detect contention for latches
- how to reduce contention for latches

Space in the Redo Log Buffer

When LGWR writes redo entries from the redo log buffer to a redo log file, user processes can then copy new entries over the entries that have been written to disk. LGWR normally writes fast enough to ensure that space is always available in the buffer for new entries, even when access to the redo log is heavy.

The statistic *redo log space requests* reflects the number of times a user process waits for space in the redo log buffer. This statistic is available through the dynamic performance table V\$SYSSTAT. By default, this table is only available to the user SYS and to users granted SELECT ANY TABLE system privilege, such as SYSTEM. Monitor this statistic over a period of time while your application is running with this query:

```
SELECT name, value
FROM v$sysstat
WHERE name = 'redo log space requests'
```

The value of *redo log space requests* should be near 0. If this value increments consistently, processes have had to wait for space in the buffer. In this case, increase the size of the redo log buffer. The size of the redo log buffer is determined by the initialization parameter LOG_BUFFER. The value of this parameter is expressed in bytes. Try increasing the size of the redo log buffer by increments of 5% until the value of *redo log space requests* nears 0.

Redo Log Buffer Latches

Access to the redo log buffer is regulated by latches. Two types of latches control access to the redo log buffer:

- the redo allocation latch
- redo copy latches

The Redo Allocation Latch

The *redo allocation latch* controls the allocation of space for redo entries in the redo log buffer. To allocate space in the buffer, an ORACLE user process must obtain the redo allocation latch. Since there is only one redo allocation latch, only one user process can allocate space in the buffer at a time. The single redo allocation latch enforces the sequential nature of the entries in the buffer.

After allocating space for a redo entry, the user process may copy the entry into the buffer while holding the redo allocation latch. Such a copy is referred to as “copying on the redo allocation latch.” A process may only copy on the redo allocation latch if the redo entry is smaller than a threshold size. After copying on the redo allocation latch, the user process releases the latch.



The maximum size of a redo entry that can be copied on the redo allocation latch is specified by the initialization parameter `LOG_SMALL_ENTRY_MAX_SIZE`. The value of this parameter is expressed in bytes. The minimum, maximum, and default values vary depending on your operating system.

Redo Copy Latches

If the redo entry is too large to copy on the redo allocation latch, the user process must obtain a *redo copy latch* before copying the entry into the buffer. While holding a redo copy latch, the user process copies the redo entry into its allocated space in the buffer and then releases the redo copy latch.

If your computer has multiple CPUs, your redo log buffer can have multiple redo copy latches. Multiple redo copy latches allow multiple processes to copy entries to the redo log buffer concurrently. The number of redo copy latches is determined by the initialization parameter `LOG_SIMULTANEOUS_COPIES`. The default value of `LOG_SIMULTANEOUS_COPIES` is the value of the initialization parameter `CPU_COUNT`. ORACLE automatically sets the value of `CPU_COUNT` to the number of CPUs available to your ORACLE instance. Do not change the value of `CPU_COUNT`.

On single-CPU computers, there should be no redo copy latches since only one process can be active at once. In this case, all redo entries are copied on the redo allocation latch, regardless of size. ORACLE sets the value of `CPU_COUNT` to 0.

Examining Redo Log Activity

Heavy access to the redo log buffer can result in contention for redo log buffer latches. Latch contention can reduce performance. ORACLE collects statistics for the activity of all latches and stores them in the dynamic performance table V\$LATCH. By default, this table is only available to the user SYS and to other users who have SELECT ANY TABLE system privilege, such as SYSTEM.

Each row in the V\$LATCH table contains statistics for a different type of latch. The columns of the table reflect activity for different types of latch requests. The distinction between these types of requests is whether the requesting process continues to request a latch if it is unavailable:

willing-to-wait	If the latch requested with a willing-to-wait request is not available, the requesting process waits a short time and requests the latch again. The process continues waiting and requesting until the latch is available.
immediate	If the latch requested with an immediate request is not available, the requesting process does not wait, but continues processing.

These columns of the V\$LATCH table reflect willing-to-wait requests:

GETS	This column shows the number of successful willing-to-wait requests for a latch.
MISSES	This column shows the number of times an initial willing-to-wait request was unsuccessful.
SLEEPS	This column shows the number of times a process waited and requested a latch after an initial willing-to-wait request.

For example, consider the case in which a process makes a willing-to-wait request for a latch that is unavailable. The process waits and requests the latch again and the latch is still unavailable. The process waits and requests the latch a third time and acquires the latch. This activity increments the statistics in these ways:

- The GETS value increases by one, since one request for the latch (the third request) was successful.
- The MISSES value increases by one, since the initial request for the latch resulted in waiting.
- The SLEEPS value increases by two, since the process waited for the latch twice, once after the initial request and again after the second request.

These columns of the V\$LATCH table reflect immediate requests:

IMMEDIATE_GETS This column shows the number of successful immediate requests for each latch.

IMMEDIATE_MISSES This column shows the number of unsuccessful immediate requests for each latch.

Monitor the statistics for the redo allocation latch and the redo copy latches over a period of time with this query:

```
SELECT name, gets , misses, immediate_gets, immediate_misses
FROM v$latch l, v$latchname ln
WHERE ln.name IN ('redo allocation', 'redo copy')
AND ln.latch# = l.latch#
```

The output of this query might look like this:

NAME	GETS	MISSES	IMMEDIATE_GETS
IMMEDIATE_MISSES			
redo allocation	252867	83	0
redo copy	0	0	22830

From the output of the query, calculate the wait ratio for each type of request.

Contention for a latch may be affecting performance if either of these conditionals true:

- if the ratio of MISSES to GETS exceeds 1%
- if the ratio of IMMEDIATE_MISSES to the sum of IMMEDIATE_GETS and IMMEDIATE_MISSES exceeds 1%



If either of these conditions is true for a latch, try to reduce contention for that latch.

These contention thresholds are appropriate for most operating systems, though some computers with many CPUs maybe able to tolerate more contention without performance reduction.

Reducing Latch Contention

Most cases of latch contention occur when two or more ORACLE processes concurrently attempt to obtain the same latch. Latch contention rarely occurs on single-CPU computers where only a single process can be active at once.

Reducing Contention for the Redo Allocation Latch

To reduce contention for the redo allocation latch, you should minimize the time that any single process holds the latch. To reduce this time, reduce copying on the redo allocation latch. Decreasing the value of the initialization parameter `LOG_SMALL_ENTRY_MAX_SIZE` reduces the number and size of redo entries copied on the redo allocation latch.

Reducing Contention for Redo Copy Latches

On multiple-CPU computers, multiple redo copy latches allow multiple processes to copy entries to the redo log buffer concurrently. The default value of `LOG_SIMULTANEOUS_COPIES` is the value of the initialization parameter `CPU_COUNT`. ORACLE automatically sets the value of `CPU_COUNT` to the number of CPUs available to your ORACLE instance. On single-CPU computers, ORACLE sets the value of `CPU_COUNT` to 0.

If you observe contention for redo copy latches, add more latches. To increase the number of redo copy latches, increase the value of `LOG_SIMULTANEOUS_COPIES`. It can help to have up to twice as many redo copy latches as CPUs available to your ORACLE instance.

Another way to reduce contention for redo copy latches is to reduce the time each process holds a latch. You can reduce this time by forcing the ORACLE user process to “pre-build” redo entries before obtaining a redo copy latch.

A redo entry may consist of many pieces. Usually each piece of a redo entry is copied individually from the user process memory into the redo log buffer while a redo copy latch is held. Copying the pieces of a redo entry individually requires many writes to memory. However, when an entry is “pre-built,” the user process assembles all the pieces together before requesting the latch. The redo entry can then be copied to the buffer in a single write to memory.

User processes pre-build all entries that are smaller than a threshold size. This threshold size is determined by the initialization parameter `LOG_ENTRY_PREBUILD_THRESHOLD`. The value of this parameter is expressed in bytes. The default value is 0. This value means that the default behavior is to not pre-build any entries.

To force your user processes to pre-build more of their redo entries, increase the value of `LOG_ENTRY_PREBUILD_THRESHOLD`.

Assigning All ORACLE Processes Equal Priority

Many processes are involved in the operation of ORACLE. These processes all access the shared memory resources in the SGA.

Be sure that all ORACLE processes, both background processes and user processes, have the same process priority. When you install ORACLE, all background processes are given the default priority for your operating system. You should not change the priorities of background processes. You should also be sure that all user processes have the default operating system priority.

Assigning different priorities to ORACLE processes may exacerbate the effects of contention. Your operating system may not grant processing time to a low priority process if a high priority process also requests processing time. If a high priority process needs access to a memory resource held by a low priority process, the high priority process may wait indefinitely for the low priority process to obtain the CPU, process, and release the resource.

ADDITIONAL TUNING CONSIDERATIONS

This chapter discusses additional tuning measures you should take after completing the steps presented in the first six chapters of this Guide. These measures tune ORACLE operations that only reduce performance in very special cases. The topics covered by this chapter include:

- sorts
- free lists
- checkpoints

Tuning Sorts

Some applications cause ORACLE to sort data. This section tells you:

- how to allocate memory to optimize your sorts
- how to avoid sorting when creating indexes

Allocating Memory for Sort Areas



Concepts

The default sort area size is adequate to hold all the data for most sorts. However, if your application often performs large sorts on data that does not fit into the sort area, then you may want to increase the sort area size. Large sorts can be caused by any SQL statement that performs a sort that operates on a large number of rows. SQL statements that perform sorts are listed in Chapter 9 “Memory Structures and Processes” of the *ORACLE7 Server Concepts Manual*.

Recognizing Large Sorts

ORACLE collects statistics that reflect sort activity and stores them in the dynamic performance tables V\$SYSSTAT. By default, this table is only available to the user SYS and to users granted SELECT ANY TABLE system privilege. These statistics reflect sort behavior:

<i>sorts(memory)</i>	The value of this statistic is the number of sorts small enough to be performed entirely in sort areas without I/O to temporary segments on disk.
<i>sorts(disk)</i>	The value of this statistic is the number of sorts too large to be performed entirely in the sort area requiring I/O to temporary segments on disk.

Monitor these statistics over a period of time while your application is running with this query:

```
SELECT name, value
FROM v$sysstat
WHERE name IN ('sorts(memory)', 'sorts(disk)')
```

The output of this query might look like this:

NAME	VALUE
sorts(memory)	965
sorts(disk)	8

Increasing Sort Area Size



IUG

If a significant number of sorts require disk I/O to temporary segments, then your application's performance may benefit from increasing the size of the sort area. In this case, increase the value of the initialization parameter SORT_AREA_SIZE. The maximum value of this parameter varies depending on your operating system.

Performance Benefits of Large Sort Areas

Increasing the size of the sort area increases the size of each run and decreases the total number of runs. This decrease can improve performance in two ways:

- Reducing the total number of runs reduces the number of merges ORACLE must perform to obtain the final sorted result.
- Reducing the total number of runs reduces the disk I/O to the temporary segment necessary to perform the sort.

Performance Tradeoffs for Large Sort Areas

Increasing the size of the sort area causes each ORACLE process that sorts to allocate more memory. This increase reduces the amount of memory available for private SQL and PL/SQL areas. It can also affect operating system memory allocation and may induce paging and swapping. Before increasing the size of the sort area, be sure enough free memory is available on your operating system to accommodate a larger sort area.

If you increase the size of your sort area, you may consider decreasing the retained size of the sort area, or the size to which ORACLE reduces the sort area if its data is not expected to be referenced soon. To decrease the retained size of the sort area, decrease the value of the initialization parameter `SORT_AREA_RETAINED_SIZE`. A smaller retained sort area reduces memory usage but causes additional I/O to write and read data to and from temporary segments on disk.

Avoiding Sorts

One cause of sorting is the creation of indexes. Creating an index for a table involves sorting all the rows in the table based on the values of the indexed column or columns.

ORACLE also allows you to create indexes without sorting. If the rows in the table are loaded in ascending order, you can create the index faster without sorting.

The NOSORT Option

To create an index without sorting, load the rows into the table in ascending order of the indexed column values. Your operating system may provide you with a sorting utility to sort the rows before you load them.

When you create the index, use the NOSORT option on the CREATE INDEX command. For example, this CREATE INDEX statement creates the index EMP_INDEX on the ENAME column of the EMP table without sorting the rows in the EMP table:

```
CREATE INDEX emp_index  
ON emp(ename)  
NOSORT
```

Choosing When to Use the NOSORT Option

Presorting your data and loading it in order may not always be the fastest way to load a table. If you have a multiple-CPU computer, you may be able to load data faster using multiple processors in parallel, each processor loading a different portion of the data. To take advantage of parallel processing, load the data without sorting it first. Then create the index without the NOSORT option.

On the other hand, if you have a single-CPU computer, you should sort your data before loading, if possible. Then you should create the index with the NOSORT option.

Reducing Free List Contention

Free list contention can reduce the performance of some applications. This section tells you:

- how to identify contention for free lists
- how to increase the number of free lists

Identifying Free List Contention

Contention for free lists is reflected by contention for free data blocks in the buffer cache. You can determine whether contention for free lists is reducing performance by querying the dynamic performance table V\$WAITSTAT.

The V\$WAITSTAT table contains block contention statistics. By default, this table is only available to the user SYS and to other users who have SELECT ANY TABLE system privilege, such as SYSTEM. The free list statistic reflects contention for free blocks. Monitor this statistic over a period of time while your application is running with this query:

```
SELECT class, count
FROM v$waitstat
WHERE class = 'free list'
```

The result of this query might look like this:

CLASS	COUNT
free list	459

Compare the number of waits for free blocks with the total number of requests for data over the same period of time. You can monitor the total number of requests for data over a period of time with this query:

```
SELECT SUM(value)
FROM v$sysstat
WHERE name IN ('db block gets', 'consistent gets')
```

The output of this query might look like this:

SUM(VALUE)
929530

If the number of waits for free blocks is greater than 1% of the total number of requests, you should consider adding more free lists to reduce contention.

Adding More Free Lists To reduce contention for the free lists of a table, re-create the table with a larger value for the FREELISTS storage parameter. Increasing the value of this parameter to the number of ORACLE processes that concurrently insert data into the table may benefit performance for the INSERT statements.

Re-creating the table may simply involve dropping and creating it again. However, you may want to use one of these means instead:

- Re-create the table by selecting data from the old table into a new table, dropping the old table, and renaming the new one.
- Use Import and Export to export the table, drop the table, and import the table. This measure avoids consuming space by creating a temporary table.

Tuning Checkpoints

A checkpoint is an operation that ORACLE performs automatically. Checkpoints can momentarily reduce performance. This section tells you:

- how checkpoints affect performance
- how to choose the frequency of checkpoints
- how to reduce the performance impact of a checkpoint

How Checkpoints Affect Performance

Checkpoints affect:

- recovery time performance
- runtime performance

Recovery Time Performance

Frequent checkpoints can reduce recovery time in the event of an instance failure. If checkpoints are relatively frequent, then relatively few changes to the database are made between checkpoints. In this case, relatively few changes must be rolled forward for recovery.

Runtime Performance

A checkpoint can momentarily reduce runtime performance for these reasons:

- Checkpoints cause DBWR to perform I/O.
- If CKPT is not enabled, checkpoints cause LGWR to update data files and may momentarily prevent LGWR from writing redo entries.

The overhead associated with a checkpoint is usually small and only affects performance while ORACLE performs the checkpoint.

Choosing Checkpoint Frequency

You should choose a checkpoint frequency based on your performance concerns. If you are more concerned with efficient runtime performance than recovery time, choose a lower checkpoint frequency. If you are more concerned with fast recovery time than runtime performance, choose a higher checkpoint frequency.

Because checkpoints on log switches are necessary for redo log maintenance, you cannot eliminate checkpoints entirely. However, you can reduce checkpoint frequency to a minimum by setting these parameters:

- Set the value of the initialization parameter `LOG_CHECKPOINT_INTERVAL` larger than the size of your largest redo log file.
- Set the value of the initialization Parameter `LOG_CHECKPOINT_TIMEOUT` to 0. This value eliminates time-based checkpoints.

Such settings eliminate all checkpoints except those that occur on log switches.

You can further reduce checkpoints by reducing the frequency of log switches. To reduce log switches, increase the size of your redo log files so that the files do not fill as quickly.

Reducing the Performance Impact of a Checkpoint

You may notice a momentary drop in performance as a checkpoint completes. This drop may be due to an accumulation of redo entries in the redo log buffer. LGWR maybe too busy updating data file headers to write these entries to a redo log file. In this case, you can reduce the performance impact of a checkpoint by enabling the Checkpoint process (CKPT).

CKPT updates data file headers when a checkpoint occurs, leaving LGWR free to write redo entries. To enable CKPT, set the value of the initialization parameter `CHECKPOINT_PROCESS` to `TRUE`. To disable CKPT, set this value to `FALSE`. CKPT is disabled by default. Before enabling CKPT, be sure that your operating system can support an additional process. You may also need to increase the value of the initialization parameter `PROCESSES`.

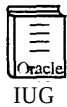
Reference

A

INITIALIZATION
PARAMETER FILES

This appendix contains detailed descriptions of the database initialization parameters. The initialization parameters are found in the parameter file. A parameter file is required to start an instance.

Though the usual name of the parameter file is INIT.ORA, the actual name of the file may vary depending on the distribution media for your operating system. For example, it maybe in mixed or lowercase, or it may have a logical name or a variation on the name INIT.ORA. As the DBA, you can choose a different filename for your parameter file.



The expected location (where the Server expects to find the parameter file upon startup) is documented in the installation or user's guide for your operating system, along with the original name of the file.

A sample parameter file is included on the ORACLE Server distribution media for each operating system. The distributed sample file is appropriate for initial use, but you will want to make changes in the file to tune your database system for best performance. The changes take effect the next time the instance is started after being completely shutdown.

Database administrators can use initialization parameters to:

- optimize performance by adjusting memory structures (for example, the number of database buffers in memory)
- set some database-wide defaults (for example, how much space is initially allocated for a context area when it is created)
- set database limits (for example, the maximum number of database users)
- specify names of files

Many initialization parameters can be “fine-tuned” to improve database performance. Other parameters should never be altered or only be altered under supervision by Oracle Corporation personnel.

Specifying Values in the Parameter File

The parameter file is a text file that contains a list of parameters and a value for each parameter, as in:

```
SAVEPOINTS = 5  
PROCESSES = 100
```

The following rules govern the specification of parameters in the parameter file:

- All parameters are optional.
- Only parameters and comments should appear in the parameter file.
- For each parameter, a default value is built into the server. This value may be operating system dependent, depending on the parameter.
- Parameters can be specified in any order.
- Case (upper or lower) is only significant (in filenames) if case is significant on the host operating system. Refer to your installation or user’s guide if you require more information.
- To enter several parameters on one line, use spaces between parameter names, as in:

```
PROCESSES = 100 SAVEPOINTS = 5 OPEN_CURSORS = 10
```
- For some parameters (such as INIT_SQL_FILES), multiple values can be entered. Multiple values must be separated by commas and enclosing them in parentheses, as in:

```
INIT_SQL_FILES = (SQL.BSQ, CATALOG.SQL, ENROLL_USERS.SQL)
```



- A backslash (\) can be used to indicate continuation of the parameter specification. If a backslash is used to continue a line, the continued line must not have any preceding spaces. For example:

```
ROLLBACK_SEGMENTS = (SEG1, SEG2, \
SEG3, SEG4, SEG5)
```

- The keyword IFILE can be used to embed another file, which is expected to be in the same format as the original parameter file. Up to three levels of nesting are allowed.
- A pound sign (#) can be used to start a comment; the rest of the line is ignored.
- Double quotes can surround parameter values, such as file names. This syntax option is useful for values that contain special characters.



Tip: It is a good idea to keep parameters in alphabetical order in the control file. That makes it easier to find them and helps to ensure that each parameter is specified only once. (If a parameter is specified more than once and you change its value, then the old value may continue to be used.)

Changing Parameter Values

To change a parameter's value, edit the parameter file. The next time the instance starts it uses the new parameter values in the updated parameter file. *Note that the change does not take effect until the **next** time the instance is started.*

Displaying Current Parameter Values

To see the current settings for initialization parameters, use the SQL*DBA command:

```
SQLDBA> SHOW PARAMETERS
```

This displays all parameters in alphabetical order, with their current values. If you enter a text string, as in:

```
SQLDBA> SHOW PARAMETERS BLOCK
```

you will see a display for all parameters having BLOCK in their name. If you display all the parameters, you may wish to use the SPOOL command to write the output to a file.

You can also see the values by including the keyword LIST as the first line in the parameter file. Then when you start an instance using that parameter file, you see a display showing the current settings for the parameters.

Groups of Parameters

Initialization parameters can be grouped in several different ways. For example, there are parameters that

- set database-wide limits
- set user or process limits
- name files or directories required by a database system
- set limits on database resources
- affect performance (these are called *variable parameters*)

Perhaps the group of parameters of most interest to database administrators is the set of variable parameters, used primarily for improving database performance.

Some parameters may never have to be specified in the parameter file:

- parameters that you never alter except when instructed to do so by Oracle Corporation to resolve a problem
- *derived parameters*, which usually should not be altered because their values are automatically calculated by ORACLE

Variable Parameters

The variable initialization parameters offer the most potential for improving system performance. Some variable parameters set capacity limits but do not affect performance. For example, when the value of OPEN_CURSORS is 10, a user process attempting to open its 11th cursor receives an error. Other variable parameters affect performance but do not impose absolute limits. For example, reducing the value of DB_BLOCK_BUFFERS does not prevent work even though it may slow down performance.

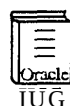
Increasing the values of variable parameters may improve your system's performance, but increasing most parameters also increases the SGA size. A larger SGA can improve database performance up to a point. In virtual memory operating systems, an SGA that is too large can degrade performance if it is swapped in and out of memory. Operating system parameters that control virtual memory working areas should be set with the SGA size in mind. The operating system configuration can also limit the maximum size of the SGA.

Global Constants
Parameters (with
Prefix “GC”)



Initialization parameters with the prefix “GC” (such as GC_DB_LOCKS) apply to systems using the ORACLE Parallel Server. The prefix “GC” stands for Global Constants. The settings of these parameters determine how the ORACLE Parallel Server coordinates multiple instances. The settings you choose have an effect on the use of certain operating system resources. For more information, refer to the *ORACLE7 Parallel Server Administrator’s Guide*, your installation or user’s guide, system release bulletins, or other operating system specific information.

Operating System
Dependent Parameters



For some initialization parameters, the valid values or ranges depend upon the host operating system; this is denoted in the default or range column as O/S dependent. For example, the parameter DB_BLOCK_BUFFERS indicates the *number* of data buffers in main memory, and its maximum value depends on the operating system; the *size* of those buffers, set by DB_BLOCK_SIZE, has a system-dependent default value. Refer to your installation and user’s guide for operating-system-specific information on the initialization parameters.

Derived Parameters

Some initialization parameters are noted as *derived*. This means that their values are calculated from the values of other parameters. Normally, you should not alter values for derived parameters but if you do, the value you specify overrides the calculated value.

**When Parameters Are
Set Incorrectly**

Some parameters have a minimum setting below which an ORACLE instance will not start; whereas if the values are too low (or high) for other parameters, ORACLE may perform badly but will still run.

You may see messages indicating that a parameter is too low or too high, or that you have reached the maximum for some resource. Frequently, you can wait a short while and retry the operation when the system is not as busy. If a message occurs repeatedly, you should shut down the instance, adjust the relevant parameter, and restart the instance.

Special Symbols and Keywords

The following special symbols and keywords control the processing of the initialization parameter file.

Used to signify a comment.

**** Used to signify that a continuation line follows on the next line. If a backslash is used to continue a line, the continued line must not have any preceding spaces. For example:

```
ROLLBACK_SEGMENTS = (SEG1, SEG2, \
SEG3, SEG4, SEG5)
```

IFILE

Default value: null

Range of values: valid parameter file names

Multiple instances: can have different values

Embeds another parameter file within the current parameter file. For example:

```
IFILE = COMMON.ORA
```

You can have up to three levels of nesting. In this example, the file COMMON.ORA could contain a second IFILE parameter for the file COMMON2.ORA, which could contain a third IFILE parameter for the file GCPARMS.ORA. You can also include multiple parameter files in one parameter file by listing IFILE several times with different values:

```
IFILE = DBPARMS.ORA
```

```
IFILE = GCPARMS.ORA
```

```
IFILE = LOGPARMS.ORA
```

For more information: Chapter 6 “Managing Control Files”

LIST

If this keyword is included, current parameter values display upon system startup. The LIST keyword must be the first line of the parameter file or an error occurs.

Reading the Parameter Descriptions

The parameter descriptions in this appendix follow the format shown below.

PARAMETER_NAME	Default value:	the value this parameter will assume if not explicitly specified.
	Range of values:	the valid range of values that this parameter can assume, shown as a minimum and maximum value. Not applicable to all parameters.
	Multiple instances:	how the values for this parameter must be specified for multiple instances in an ORACLE parallel server. Not applicable to all parameters.
The remaining paragraphs provide a textual description of the parameter and the effects of different settings.		
For more information: references to chapters or books that contain more detailed information on this subject.		

Parameter Descriptions



Descriptions of the individual initialization parameters follow in alphabetical order. Parameter names are shown in uppercase for readability. Refer to the installation or user's guide to see how your operating system expects to read the file and the parameters.

Most initialization parameter values are global (on a database-wide basis), not per user, unless otherwise specified.

AUDIT_TRAIL

Default value: NONE

Range of values: NONE, DB, OS

Enables or disables the writing of rows to the audit trail. Auditing records are not written if the value is NONE or if the parameter is not present. The OS option enables system-wide auditing and causes auditing records to be written to the operating system's audit trail. The DB option enables system-wide auditing and causes audit records to be written to the database audit trail (the SYS.AUD\$ table).

The values TRUE and FALSE are also supported, for backward compatibility. TRUE is equivalent to DB. FALSE is equivalent to NONE.

The SQL AUDIT statements can be used to set auditing options regardless of the setting of this parameter.

For more information: Chapter 13 "Auditing Database Use"

AUTO_MOUNTING

This is a Trusted ORACLE parameter.



For more information: *Trusted ORACLE7 Server Administrator's Guide*

BACKGROUND_DUMP_DEST

Default value: O/S dependent

Range of values: valid local pathname, directory, or disk

The path name for a directory where debugging trace files for the background processes (LGWR, DBWR, etc.) are written during the course of ORACLE operations.

An ALERT file in the directory specified by BACKGROUND_DUMP_DEST logs significant database events and messages. This file records all instance startups and shutdowns, messages to the operator console, and errors that cause trace files to be written. It also records every CREATE, ALTER, or DROP operation on a database, tablespace, or rollback segment.



The ALERT file is a normal text file. Its filename is operating system specific, generally of the form *nodename_instance-name_ALERT.LOG*. This file grows slowly, but without limit, so the DBA may wish to delete it periodically. The file may be deleted even when the database is running.

For more information: Chapter 4 “Managing ORACLE Processes”

CHECKPOINT_PROCESS

Default value: false

Range of values: true/false

Multiple instances: can have different values

Setting this parameter to TRUE enables the CKPT background process. You should only enable the CKPT process if the performance of the LGWR process decreases significantly during a checkpoint.

Note: Adjust all calculations that depend on the number of background processes to allow for the CKPT process. For example, increase the value of the PROCESSES parameter by one, and increase the values of other parameters whose default values are derived from PROCESSES if you do not use their default values.

For more information: see PROCESSES on page A-39

CLEANUP_ROLLBACK_ENTRIES

Default value: 20 entries

The number of undo records processed at one time when rolling back a transaction. Prevents long transactions from freezing out shorter transactions that also need to be rolled back. Normally this parameter will not need modification.

For more information: Chapter 9 “Managing Rollback Segments”

COMMIT_POINT_STRENGTH

Default value: O/S dependent

Range of values: 0..255

A value used to determine the commit point site in a distributed transaction. The node in the transaction with the highest value for COMMIT_POINT_STRENGTH will be the commit point site. A database’s commit point strength should be set relative to the amount of critical shared data in the database. For example, a database on a mainframe computer typically shares more data among users than on a personal computer. Therefore, COMMIT_POINT_STRENGTH should be set to a higher value for the mainframe computer.

The commit point site stores information about the status of transactions. Other computers in a distributed transaction require this information, so it is desirable to have a machines that are always available as commit point sites. Therefore, set COMMIT_POINT_STRENGTH to a higher value on your more available machines.



For more information: *ORACLE7 Server Concepts Manual*, Chapter 22 “Two-Phase Commit” See you installation or users guide for the default value.

COMPATIBLE

Default value: Release Dependent
Range of values: Default Version to Current Version
Multiple instances: Must have the same value

This parameter allows a new release to be used while at the same time guaranteeing backwards compatibility with an earlier version (in case it becomes necessary to retreat to the earlier version). The version that the ORACLE server must remain compatible with is specified by this parameter. In order to maintain compatibility, some features of the current version may be restricted.

The default value is the earliest version with which compatibility can be guaranteed.

For more information: For the default value, see your ORACLE server release notes. For more information, see the *ORACLE7 Server Migration Guide*.

COMPATIBLE_NO_RECOVERY

Default value: Release Dependent
Range of values: Default version to current version
Multiple instances: Must have the same value

This parameter functions like the Compatible parameter, except that the earlier version may not be usable on the current database if any recovery is needed.

The default value is the earliest version with which compatibility can be guaranteed. In some cases, this version maybe earlier than the version specifiable with the COMPATIBLE parameter.

For more information: For the default value, see your ORACLE server release notes. For more information, see the *ORACLE7 Server Migration Guide*.

CONTROL_FILES

Default value: O/S dependent
Range of values: 1..8 filenames

One or more names of control files, separated by commas. Oracle Corporation recommends using multiple files on different devices.

For more information: Chapter 6 “Managing Control Files”

CURSOR_SPACE_FOR_TIME

Default value: false
Range of values: true/false

Setting this parameter to TRUE causes the database to use more space for cursors in order to save time. It affects both the shared SQL area and the client's private SQL area.

Shared SQL areas are kept pinned in the shared pool when this parameter's value is TRUE. As a result, shared SQL areas are not aged out of the pool as long as there is an open cursor that references them. Since each active cursor's SQL area is present in memory, execution is faster. However, since the shared SQL areas never leave memory while they are in use, you should only set this parameter to TRUE when the shared pool is large enough to hold all cursors simultaneously.

Setting this parameter to TRUE also retains the private SQL area allocated for each cursor between executes instead of discarding it after cursor execution. This setting saves cursor allocation and initialization time.

For more information: *ORACLE7 Server Concepts Manual*, Chapter 11
"SQL and PL/SQL"

DB_BLOCK_BUFFERS

Default value: 32 buffers
Range of values: 4..unlimited

The number of database blocks cached in memory in the SGA (one block equals one buffer). This parameter is the most significant determinant of the SGA size and database performance. The advantage of a higher value is that when a user needs a database block that block is more likely to be in memory, thus reducing I/O. The disadvantage of high values is that more memory is consumed. The size of each buffer is equal to the size of the parameter DB_BLOCK_SIZE.

For more information: *ORACLE7 Server Concepts Manual*, Chapter 9
"Memory Structures and Processes"

DB_BLOCK_CHECKPOINT_BATCH

Default value: derived (DB_BLOCK_WRITE_BATCH / 4)
Range of values: 0..DB_BLOCK_WRITE_BATCH

The maximum number of blocks that the database writer process will write in one batch that are devoted to checkpoints. The maximum size of the write-batch is set by DB_BLOCK_WRITE_BATCH. Setting DB_BLOCK_CHECKPOINT_BATCH to a small value, relative to the write-batch size, prevents the I/O system from being flooded with checkpoint writes and allows other modified blocks to be written to disk. Setting it to a higher value allows checkpoints to complete more quickly.

In general, DB_BLOCK_CHECKPOINT_BATCH should be set to a value that allows the checkpoint to complete before the next checkpoint log switch takes place. If a checkpoint log switch takes place every 20 minutes, then this parameter should be set to a value that allows checkpointing to complete within 20 minutes.

setting DB_BLOCK_CHECKPOINT_BATCH to zero causes the default value to be used. A value larger than DB_BLOCK_WRITE_BATCH can be specified, but its effect is the same as specifying the maximum.

For more information: *ORACLE7 Server Concepts Manual*, Chapter 9
“Memory Structures and Processes”

DB_BLOCK_LRU_EXTENDED_STATISTICS

Default value: 0
Range of values: 0..unlimited

Disables or enables compilation of statistics in the X\$KCBRBH table, which measures the effects of increasing the number of buffers in the buffer cache in the SGA. When this facility is enabled, it keeps track of the number of disk accesses that would be saved if additional buffers were allocated. A value greater than zero specifies the additional number of buffers (over DB_BLOCK_BUFFERS) for which statistics are kept. This tuning tool should be turned off during normal operation.

When compiling statistics for the X\$KCBRBH table, set this parameter to the maximum size you want to use to evaluate the buffer cache. It should be set to 0 otherwise.

For more information: Chapter 21 “Tuning Memory Allocation”

DB_BLOCK_LRU_STATISTICS

Default value: false
Range of values: true/false

Disables or enables compilation of statistics in the X\$KCBCBH table, which measures the effect of fewer buffers in the buffer cache in the SGA. This tuning tool should be turned off during normal operation.

Set this parameter to TRUE when you want to compile statistics for the X\$KCBCBH table; otherwise, leave it set to FALSE.

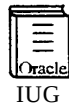
For more information: Chapter 21 “Tuning Memory Allocation”

DB_BLOCK_SIZE

Default value: O/S dependent
Range of values: O/S dependent (1024 - 8192)
Multiple instances: must have the same value

The size in bytes of ORACLE database blocks. Typical values are 2048 and 4096. The value for DB_BLOCK_SIZE in effect at CREATE DATABASE time determines the size of the blocks; at all other times the value must be set to the original value.

This parameter affects the maximum value of the FREELISTS storage parameter for tables and indexes.



For more information: *ORACLE7 Server Concepts Manual*, Chapter 3 “Database Blocks, Extents, and Segments” (block size). See your installation or user’s guide for the default value.

DB_DOMAIN

Default value: WORLD
Range of values: any legal string of name components, separated by periods and up to 128 characters long, including periods
Multiple instances: must have the same value

The extension components of a global database name, consisting of valid identifiers, separated by periods. Specifying DB_DOMAIN as a unique string for every database is highly recommended.

For example, this parameter allows one department to create a database without worrying that it might have the same name as a database created by another department. If one sales department’s DB_DOMAIN = “JAPAN.ACME.COM”, then their “SALES” database is uniquely distinguished from another department that might also have DB_NAME = “SALES” but with DB_DOMAIN = “US.ACME.COM”.

For more information: Chapter 15 “Managing Distributed Databases”

DB_FILES

Default value: O/S dependent
Range of values: minimum: MAXDATAFILES for the database to be mounted
maximum: O/S dependent
Multiple instances: must have the same value

The maximum number of database files that can be opened at runtime for this database. If you increase the value, you must shut down and restart all instances accessing the database before the new value can take effect.

Reduce the value only if you need SGA space and do not anticipate having more database files.

DB_FILES is similar to the MAXDATAFILES argument for the CREATE DATABASE statement, which sets the absolute maximum number of data files at database creation. An instance cannot mount a database unless DB_FILES is equal to or greater than MAXDATAFILES for that database. Greater values are only useful for instances that mount different databases at different times.



For more information: Chapter 7 “Managing the Size and Files of the Database” See your installation or user’s guide for the default value.

DB_FILE_MULTIBLOCK_READ_COUNT

Default value: O/S dependent
Range of values: O/S dependent

Used for multi-block I/O, this is the maximum number of blocks read in one I/O operation during a sequential scan. The default is a function of DB_BLOCK_BUFFERS and PROCESSES. Values in the range of 4 to 16 or even 32 are reasonable. The actual maximums vary by operating system.



For more information: *ORACLE7 Server Concepts Manual*, Chapter 13 “The Optimizer” See your installation or user’s guide for the default value and range of values.

DB_FILE_SIMULTANEOUS_WRITES

Default value: 4
Range of values: 1..24

The number of simultaneous writes (“batches”) for each database file when written by DBWR.

If the operating system only supports one write per device and cannot combine writes to adjacent blocks, then the value should be 1. Though the value has no maximum because DBWR writes blocks in groups, it is not useful to use a value larger than that value. Refer to your installation or user’s guide for information specific to your operating system.



IUG

For more information: *ORACLE7 Server Concepts Manual*, Chapter 9
“Memory Structures and Processes”

DB_MOUNT_MODE

This is a Trusted ORACLE parameter.



Trusted

For more information: *Trusted ORACLE7 Server Administrator’s Guide*

DB_NAME

Default value: null
Range of values: Any valid database name
Multiple instances: must have the same value, or else the same value must be specified in `STARTUP OPEN db_name` or `ALTER DATABASE db_name MOUNT`.

A database identifier of up to eight characters. If specified, it must correspond to the name specified in the `CREATE DATABASE` statement. Although the use of `DB_NAME` is optional, it should generally be set before invoking `CREATE DATABASE` and then referenced in that statement.

If not specified, a database name must appear on either the `STARTUP` or the `ALTER DATABASE MOUNT` command line.

The following are valid characters in a database name:

- alphabetic characters
- numbers
- underscore (`_`)
- sharp (`#`)
- dollar sign (`$`)

No other characters are valid. Double quotes are removed before processing the database name. They cannot be used to embed other characters in the name.

Lowercase characters are not treated with any special significance. They are considered the same as their uppercase counterparts.

For more information: `DB_DOMAIN`, page A-14, Chapter 15 “Managing Distributed Databases”, and Chapter 2 “Creating a Database”

DISCRETE_TRANSACTIONS_ENABLED

Default value: false
Range of values: true/false

Implements a simpler, faster rollback mechanism that improves performance for certain kinds of transactions. There are strict limits on the kinds of transactions that can occur in discrete mode, but greater efficiency can be obtained for these transactions.

For more information: *ORACLE7 Server Application Developer's Guide*, Appendix A “Supplied Packages”

DISTRIBUTED_LOCK_TIMEOUT

Default value: 60 seconds
Range of values: 1..unlimited

The amount of time in seconds for distributed transactions to wait for locked resources.

For more information: *ORACLE7 Server Concepts Manual*, Chapter 10 “Data Concurrency” and Chapter 22 “Two-Phase Commit”

DISTRIBUTED_RECOVERY_CONNECTION_HOLD_TIME

Default value: 200 seconds
Range of values: 0..1800 seconds

The length of time to hold a remote connection open after a distributed transaction fails, in hopes that communication will be restored without having to reestablish the connection. Larger values minimize reconnection time, but they also consume local resources for a longer time period. Values larger than 1800 seconds can be specified, but since the reconnection and recovery background process runs every 30 minutes (1800 seconds) whether or not a failure occurs, any value of 1800 or larger means that the connection never closes.

For more information: Chapter 15 “Managing Distributed Databases”

DISTRIBUTED_TRANSACTIONS

Default value: O/S dependent
Range of values: 0..TRANSACTIONS

The maximum number of distributed transactions that this database can concurrently participate in. The value of this parameter cannot exceed the value of the parameter TRANSACTIONS.

If network failures are occurring at an abnormally high rate, causing many in-doubt transactions, you may want to temporarily decrease this parameter's value. This limits the number of concurrent distributed transactions, which then reduces the number of in-doubt transactions. Thus, the amount of blocked data and possible heuristic decision making (because of in-doubt transactions) is reduced.

If DISTRIBUTED_TRANSACTIONS is set to 0, no distributed transactions are allowed for the database and the recoverer (RECO) process does not start when the instance starts up.



For more information: Chapter 15 “Managing Distributed Databases”. See your installation or user's guide for the default value.

DML_LOCKS

Default value: derived (4*TRANSACTIONS)
Range of values: 20..unlimited, 0
Multiple instances: must all have positive values or must all be 0

The maximum number of DML locks — one for each table modified in a transaction. Value should equal the grand total of locks on tables referenced by all users. For example, if 3 users are modifying data in one table, then 3 entries would be required. If 3 users are modifying data in 2 tables, then 6 entries would be required.

The default value assumes an average of 4 tables referenced per transaction. For some systems, this value may not be enough.

If the value is set to 0, enqueues are disabled and performance is slightly increased. However, you cannot use DROP TABLE, CREATE INDEX, or explicit lock statements such as LOCK TABLE IN EXCLUSIVE MODE.

For more information: *ORACLE7 Server Concepts Manual*, Chapter 10 “Data Concurrency”

ENQUEUE_RESOURCES	Default value: derived Range of values: 10..65535
	The number of resources that can be locked by the lock manager. The default value is derived from PROCESSES and should be adequate. For three or fewer processes, the default value is 20. For 4 to 10 processes, the default value is $((\text{PROCESSES}-3)*5) + 20$; and for more than 10 processes it is $((\text{PROCESSES}-10)*2) + 55$. If many tables are used, the value maybe increased. Allow one per resource (regardless of the number of sessions or cursors using that resource), not one per lock. The value should be no greater than DML_LOCKS plus DDL_LOCKS plus overhead of about 20, and generally should be much less. For more information: <i>ORACLE7 Server Concepts Manual</i>, Chapter 10 “Data Concurrency”
EVENT	Default value: null Modifies the scope of ALTER SESSION SET EVENTS commands so that they pertain to an entire instance, instead of to a single session. Used for system debugging. This parameter should not usually be altered except at the direction of Oracle personnel.
FIXED_DATE	Default value: null Allows you to set a constant for SYSDATE in the format ‘YYYY-MM-DD-HH24:MI:SS’. Also accepts the default ORACLE date format, without a time. Useful primarily for testing. For more information: <i>ORACLE7 Server Concepts Manual</i>, Chapter 6 “Datatypes”
GC_DB_LOCKS	This is a parallel server parameter. For more information: <i>ORACLE7 Parallel Server Administrator’s Guide</i>
GC_FILES_TO_LOCKS	This is a parallel server parameter. For more information: <i>ORACLE7 Parallel Server Administrator’s Guide</i>
GC_LCK_PROCS	This is a parallel server parameter. For more information: <i>ORACLE7 Parallel Server Administrator’s Guide</i>

GC_ROLLBACK_LOCKS	This is a parallel server parameter. For more information: <i>ORACLE7 Parallel Server Administrator's Guide</i>
GC_ROLLBACK_SEGMENTS	This is a parallel server parameter. For more information: <i>ORACLE7 Parallel Server Administrator's Guide</i>
GC_SAVE_ROLLBACK_LOCKS	This is a parallel server parameter. For more information: <i>ORACLE7 Parallel Server Administrator's Guide</i>
GC_SEGMENTS	This is a parallel server parameter. For more information: <i>ORACLE7 Parallel Server Administrator's Guide</i>
GC_TABLESPACES	This is a parallel server parameter. For more information: <i>ORACLE7 Parallel Server Administrator's Guide</i>
GLOBAL_NAMES	Default value: false Range of values: true/false This parameter determines whether or not a database link is required to have the same name as the database it connects to. If the value of GLOBAL_NAMES is false, then no check is performed. ORACLE recommends setting this parameter to TRUE to ensure the use of consistent naming conventions for databases and links. For more information: Chapter 15 “Managing Distributed Databases”

INIT_SQL_FILES

Default value: O/S dependent

Range of values: valid SQL script filenames

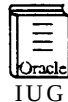
Multiple instances: can have different values

Specifies the name of one or more files containing SQL statements to be executed during database creation. Enclose the list in parentheses and separate the filenames with commas, as in:

```
INIT_SQL_FILES = (SQL.BSQ, CATALOG.ORA, SITEDD.SQL)
```

This parameter should always specify SQL.BSQ first. Then other site-specific files can be specified if desired. The default is operating system dependent and should not be altered, except to include additional files. Objects created with additional files that you add belong to the user SYS.

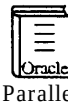
The INIT_SQL_FILES parameter has no effect except when executing the CREATE DATABASE statement. Since CREATE DATABASE can only be executed when a single instance is started up with the NOMOUNT option, this parameter does not apply to instances of an ORACLE Parallel Server running in parallel mode.



For more information: Chapter 2 “Creating a Database”. See your installation or user’s guide for the default value.

INSTANCE_NUMBER

This is a parallel server parameter.



For more information: *ORACLE7 Parallel Server Administrator’s Guide*

LABEL_CACHE_SIZE

This is a Trusted ORACLE parameter.



For more information: *Trusted ORACLE7 Server Administrator’s Guide*

LICENSE_MAX_SESSIONS	Default value: 0 Range of values: 0..number of session licenses Multiple instances: can have different values Sets the maximum number of concurrent user sessions allowed simultaneously. When this limit is reached, only users with the RESTRICTED SESSION privilege can connect to the server. Users who are able to connect receive a warning message indicating that the system has reached maximum capacity. A zero value indicates that concurrent usage (session) licensing is not enforced. If you set this parameter to a non-zero number, you may also want to set LICENSE_SESSIONS_WARNING. Concurrent usage licensing and user licensing should not both be enabled. Either LICENSE_MAX_SESSIONS or LICENSE_MAX_USERS should always be zero. Multiple instances can have different values, but the total for all instances mounting a database should be less than or equal to the total number of sessions licensed for that database. For more information: Chapter 11 “Controlling Access to the Database”
LICENSE_MAX_USERS	Default value: 0 Range of values: 0..number of user licenses Multiple instances: should have the same values Sets the maximum number of users you can create in the database. When you reach this limit, you cannot create any more users. You can, however, increase the limit. Concurrent usage (session) licensing and user licensing should not both be enabled. Either LICENSE_MAX_SESSIONS or LICENSE_MAX_USERS, or both, should be zero. If different instances specify different values for this parameter, the value of the first instance to mount the database takes precedence. For more information: Chapter 11 “Controlling Access to the Database”

LICENSE_SESSIONS_WARNING

Default value: 0
Range of values: 0..LICENSE_MAX_SESSIONS
Multiple instances : can have different values

Sets a warning limit on the number of concurrent user sessions. When this limit is reached, additional users can connect, but ORACLE writes a message in the ALERT file for each new connection. Users with RESTRICTED SESSION privilege who connect after the limit is reached receive a warning message stating that the system is nearing its maximum capacity.

If this parameter is set to zero, no warning is given when approaching the concurrent usage (session) limit. If you set this parameter to a non-zero number, you should also set LICENSE_MAX_SESSIONS.

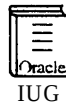
For more information: Chapter 11 “Controlling Access to the Database”

LOG_ARCHIVE_BUFFER_SIZE

Default value: O/S dependent
Range of values: 1..O/S dependent (in O/S blocks)
Multiple instances: can have different values

The size of each archival buffer, in redo log blocks (operating system blocks). The default should be adequate for most applications.

This parameter, with LOG_ARCHIVE_BUFFERS, can be used to tune archiving so that it runs as fast as necessary, but not so fast that it reduces system performance.



For more information: Chapter 17 “Archiving Redo Information”. See your installation or user’s guide for the default value and range of values.

LOG_ARCHIVE_BUFFERS **Default value:** O/S dependent
Range of values: 1..O/S dependent
Multiple instances: can have different values

The number of buffers to allocate for archiving. The default should be adequate for most applications.

This parameter, with LOG_ARCHIVE_BUFFER_SIZE, can be used to tune archiving so that it runs as fast as necessary, but not so fast that it reduces system performance.

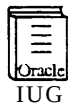


For more information: Chapter 17 “Archiving Redo Information”. See your installation or user’s guide for the default value and range of values.

LOG_ARCHIVE_DEST **Default value:** O/S dependent
Range of values: any valid path or device name
Multiple instances: can have different values

Applicable only if using the redo log in ARCHIVELOG mode. Use a text string to specify the default location and root of the disk file or tape device when archiving redo log files. (Archiving to tape is not supported on all operating systems.)

To override the destination that this parameter specifies, either specify a different destination for manual archiving or use the SQL*DBA command ARCHIVE LOG START *filespec* for automatic archiving.



See your user or installation guide for the default value and for example of how to specify the destination path or filename using LOG_ARCHIVE_DEST and LOG_ARCHIVE_FORMAT.

For more information: Chapter 17 “Archiving Redo Information”

LOG_ARCHIVE_FORMAT **Default value:** O/S dependent (length for uppercase variables is also O/S dependent)
Range of values: any valid file name
Multiple instances: can have different values, but identical values are recommended.

Applicable only if using the redo log in ARCHIVELOG mode. Use a text string and variables to specify the default filename format when archiving redo log files. The string generated from this format is appended to the string specified in the LOG_ARCHIVE_DEST parameter. The following variables can be used in the format:

%s log sequence number

%t thread number

Using uppercase letters (for example, %S) for the variables causes the value to be a fixed length padded to the left with zeros. The following is an example of specifying the archive redo log filename format:

LOG_ARCHIVE_FORMAT = 'LOG%s_%t.ARC'



For more information: Chapter 17 “Archiving Redo Information”. See your installation or user’s guide for the default value.

LOG_ARCHIVE_START **Default value:** false
Range of values: true/false
Multiple instances: can have different values

Applicable only when you use the redo log in ARCHIVELOG mode, LOG_ARCHIVE_START indicates whether archiving should be automatic or manual when the instance starts up. TRUE indicates that archiving is automatic. FALSE indicates that the DBA will archive filled redo log files manually. (The SQL*DBA command ARCHIVE LOG START or STOP overrides this parameter.)

In ARCHIVELOG mode, if all online redo log files fill without being archived, an error message is issued and instance operations are suspended until the necessary archiving is performed. This delay is more likely if you use manual archiving. You can reduce its likelihood by increasing the number of online redo log files.

To use ARCHIVELOG mode while creating a database, set this parameter to TRUE. Normally, a database is created in NOARCHIVELOG mode and then altered to ARCHIVELOG mode after creation.

For more information: Chapter 17 “Archiving Redo Information”

LOG_BUFFER

Default value: O/S dependent

Range of values: O/S dependent

The number of bytes allocated to the redo log buffer in the SGA. In general, larger values reduce redo log file I/O, particularly if transactions are long or numerous. In a busy system, the value 65536 or higher would not be unreasonable.

The default is set to 4 times the maximum database block size for the host operating system.



For more information: Chapter 23 “Tuning Contention”. See your installation or users guide for the default value and range of values.

LOG_CHECKPOINT_INTERVAL

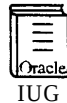
Default value: O/S dependent

Range of values: 2..unlimited (operating system blocks, not database blocks)

Multiple instances: can have different values

The number of newly filled redo log file blocks needed to trigger a checkpoint. Regardless of this value, a checkpoint always occurs when switching from one online redo log file to another. If the value exceeds the actual redo log file size, checkpoints occur only when switching logs.

The number of times DBWR has been notified to do a checkpoint for a given instance is shown in the statistic *dbwr checkpoints* displayed by the SQL*DBA command MONITOR STAT CACHE.



For more information: Chapter 24 “Additional Tuning Considerations”
See your installation or users guide for the default value.

LOG_CHECKPOINT_TIMEOUT

Default value: 0 seconds
Range of values: 0..unlimited
Multiple instances: can have different values

The amount of time to pass before another checkpoint occurs. The value is specified in seconds. A value of 0 disables time-based checkpoints. The time begins at the start of the previous checkpoint, then a checkpoint occurs after the amount of time specified by this parameter.

Note: A checkpoint scheduled to occur because of this parameter is delayed until the completion of the previous checkpoint if the previous checkpoint has not yet completed.

For more information: Chapter 24 “Additional Tuning Considerations”

LOG_FILES

Default value: 255
Range of values: 2..255 (should equal MAXLOGFILES)
Multiple instances: must have the same value

The maximum log group number. This value specifies the maximum number of redo log files that can be opened at runtime for the database. It also gives the upper limit on the group numbers that can be specified when issuing log-related commands. Reduce the value only if you need SGA space and have fewer redo log files.

The MAXLOGFILES option for the CREATE DATABASE statement sets the absolute maximum number of redo log files that can ever be created for the database. MAXLOGFILES overrides the LOG_FILES parameter value if LOG_FILES is greater than MAXLOGFILES.

For more information: Chapter 23 “Tuning Contention”

LOG_SIMULTANEOUS_COPIES

Default value: CPU_COUNT

Range of values: 0..unlimited

The maximum number of redo buffer copy latches available to write log entries simultaneously. For good performance, you can have up to twice as many redo copy latches as CPUs. For a single-processor system, set to 0 so that all log entries are copied on the redo allocation latch.

If this parameter is set to 0, redo copy latches are turned off and the parameters LOG_ENTRY_PREBUILD_THRESHOLD and LOG_SMALL_ENTRY_MAX_SIZE are ignored.

For more information: Chapter 23 “Tuning Contention”

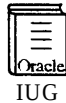
LOG_SMALL_ENTRY_MAX_SIZE

Default value: O/S dependent

Range of values: O/S dependent

The size in bytes of the largest copy to the log buffers that can occur under the redo allocation latch, without obtaining the redo buffer copy latch. If the value for LOG_SIMULTANEOUS_COPIES is 0, this parameter is ignored (all writes are “small” and are made without the copy latch).

If the redo entry is copied on the redo allocation latch, the user process releases the latch after the copy. If the redo entry is larger than this parameter, the user process releases the latch after allocating space in the buffer and getting a redo copy latch.



For more information: Chapter 23 “Tuning Contention”. See your installation or user’s guide for the default value and range of values.

MAX_DUMP_FILE_SIZE

Default value: 500 blocks

Range of values: 0..unlimited

Maximum size in operating system blocks of any trace files written. Set this limit if you are concerned that trace files may take up too much space.

For more information: Chapter 4 “Managing ORACLE Processes”

MAX_ENABLED_ROLES

Default value: 20
Range of values: 0..48

The maximum number of database roles that a user can enable, including sub-roles.

For more information: Chapter 12 “Controlling User Privileges and Roles”

MAX_ROLLBACK_SEGMENTS

Default value: 30
Range of values: 1..65536

The maximum size of the rollback segment cache in the SGA. The number specified signifies the maximum number of rollback segments that can be kept online (that is, status of INUSE) simultaneously by one instance.

For more information: Chapter 9 “Managing Rollback Segments”

MLS_LABEL_FORMAT

This is a Trusted ORACLE parameter.



Trusted

For more information: *Trusted ORACLE7 Server Administrator's Guide*

MTS_DISPATCHERS

Default value: null

The configuration of the dispatcher processes created when the instance starts up. The value of this parameter is specified as one or more configuration strings. Each configuration string is a quoted string of 2 values separated by a comma that specifies the configuration of a group of one or more dispatchers. The configuration string for each group of dispatcher processes includes the network protocol for that group and the number of dispatcher processes in the group (one or more). Each network protocol that you use on your system requires a separate specification. For example, if you are using TCP/IP and DECNet to connect to the server, you would need to specify two parameters, as follows:

```
MTS_DISPATCHERS = "tcp, 1"  
MTS_DISPATCHERS = "decnet, 4"
```

The first configuration string in this example specifies one dispatcher process for the TCP/IP protocol. The second string specifies four dispatcher processes for the DECNet protocol.

For more information: Chapter 15 “Managing Distributed Databases”

MTS_LISTENER_ADDRESS **Default value:** null

The configuration of the Listener process. The Listener process requires an address to listen for connection requests for each network protocol that is used on your system. Addresses are specified as the SQL*Net description of the connection address.



Each address must be specified with its own parameter. (This differs from the SQL*Net syntax.) For example, if you use TCP/IP as well as DECNet, you would provide specifications similar to the following in your initialization file:

```
MTS_LISTENER_ADDRESS =  
  "(ADDRESS=(PROTOCOL=tcp) (HOST=myhost) (PORT=7002))"  
  
MTS_LISTENER_ADDRESS =  
  "(ADDRESS=(PROTOCOL=decnet) (NODE=name) (OBJECT.mts))"
```



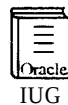
IUG

Address specifications for the Listener process are operating system specific and network protocol specific. Refer to your installation or user's guide and SQL*Net documentation for a description of how to specify addresses for the protocols on your system.

For more information: Chapter 14 “Administering Client Server Systems”

MTS_MAX_DISPATCHERS **Default value:** 5
 Range of values: O/S dependent

The maximum number of dispatcher processes allowed to be running simultaneously.



IUG

For more information: Chapter 14 “Administering Client Server Systems”. See your installation or user's guide for the range of values.

MTS_MAX_SERVERS

Default value: 20
Range of values: O/S dependent

The maximum number of shared server processes allowed to be running simultaneously.

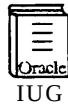


For more information: Chapter 14 “Administering Client Server Systems”. See your installation or user’s guide for the range of values.

MTS_SERVERS

Default value: 0
Range of values: O/S dependent

The number of server processes that you wish to create when an instance is started up.



For more information: Chapter 14 “Administering Client Server Systems”. See your installation or user’s guide for the range of values.

MTS_SERVICE

Default value: null

This parameter specifies the name of the service you wish to be associated with the dispatcher. Using this name in the CONNECT string allows users to connect to an instance via a dispatcher. ORACLE always checks for such a service before establishing a normal database connection.

The name you specify must be unique. It should *not* be enclosed in quotes. It is a good idea for this name to be the same as the instance name. That way, if the dispatcher is unavailable for any reason, the CONNECT string will still connect the user to the database.

If not specified, MTS_SERVICE defaults to the value specified by DB_NAME. If DB_NAME also is not specified, the ORACLE server returns an error at startup indicating that the value for this parameter is missing.

For more information: Chapter 14 “Administering Client Server Systems”

NLS_CURRENCY

Default value: derived
Range of values: any valid character string

Defines the string to use as the local currency symbol for the L number format element. The default value of this parameter is determined by NLS_TERRITORY. For example, the following query uses the L format element to return the default local currency symbol for the territory FRANCE:

```
SELECT TO_CHAR (TOTAL, 'L099') "TOTAL"
      FROM ORDERS WHERE CUSTNO = 586;

TOTAL
-----
F635
```

For more information: Appendix C “National Language Support”

NLS_DATE_FORMAT

Default value: derived
Range of values: any valid date format mask

Defines the default date format to use with the TO_CHAR and TO_DATE functions. The default value of this parameter is determined by NLS_TERRITORY. The value of this parameter can be any valid date format mask. For example:

```
NLS_DATE_FORMAT = 'MM/DD/YYYY'
```

For more information: Appendix C “National Language Support”

NLS_DATE_LANGUAGE

Default value: value for NLS_LANGUAGE
Range of values: any valid NLS_LANGUAGE value

Determines the language to use for day and month names and date abbreviations (AM, PM, AD, BC). The default value of this parameter is the language specified by NLS_LANGUAGE.

For more information: Appendix C “National Language Support”

NLS_ISO_CURRENCY

Default value: derived

Range of values: any valid NLS_TERRITORY value

Defines the string to use as the international currency symbol for the C number format element. The default value of this parameter is determined by NLS_TERRITORY. For example, the following query uses the C format element to return the default international currency symbol for the territory FRANCE:

```
SELECT TO_CHAR (TOTAL, 'C099') "TOTAL"
      FROM ORDERS WHERE CUSTNO = 586;
```

```
      TOTAL
      -----
      FRF635
```

The value of this parameter can be any valid territory specified in NLS_TERRITORY.

For more information: Appendix C “National Language Support”

NLS_LANGUAGE

Default value: O/S dependent

Defines the default language of the database. Specifies the language to use for messages, the language of day and month names, symbols to use for AD, BC, AM, and PM, and the default sorting mechanism. This parameter has the format

NLS_LANGUAGE = FRENCH

Examples of supported languages are American, French, and Japanese.

This parameter determines the default values of the parameters NLS_DATE_LANGUAGE and NLS_SORT. Also see your installation or user’s guide and your country release notes for information specific to your operating system.



For more information: Appendix C “National Language Support”, your installation or user’s guide, and your country release notes for information specific to your operating system.

NLS_NUMERIC_CHARACTERS

Default value: derived

Defines the characters to use as the group separator and decimal. The group separator is the character that separates integer groups (that is, the thousands, millions, billions, etc.). The decimal separates the integer portion of a number from the decimal portion. Any character can be used for the decimal or group separator, but both characters must be different from each other. The characters are specified in the following format

```
NLS_NUMERIC_CHARACTERS = '<decimal character><group separator>'
```

For example, if you wish to specify a comma as the decimal character and a space as the group separator, you would set this parameter as follows:

```
NLS_NUMERIC_CHARACTERS = ' , '
```

The default value of this parameter is determined by NLS_TERRITORY.

Note: When the decimal character is not a period (.) or when the group separator is used, numeric literals must appear in quotes, since the comma is used to separate items in a list in SQL. For example, with the value of NLS_NUMERIC_CHARACTERS above, the following SQL statement requires quotes around the numeric literals:

```
INSERT INTO SIZES ( ITEMID, PRICE, WIDTH )  
VALUES ( 618, '45,50', TO_NUMBER('1 234,11', '9G999D99'));
```

For more information: Appendix C “National Language Support”

NLS_SORT

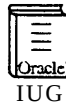
Default value: derived

Range of values: BINARY or named linguistic sort

If the value is BINARY, then the collating sequence for ORDER BY queries is based on the numeric value of characters (a binary sort that requires less system overhead).

If the value is a named linguistic sort, sorting is based on the order of the defined linguistic sort. Most languages supported by the NLS_LANGUAGE parameter also support a linguistic sort with the same name. Linguistic sorts can be created with NLS*WorkBench if your language doesn't support a linguistic sort with the appropriate sorting rules.

The default value of this parameter is dependent on the NLS_LANGUAGE parameter.



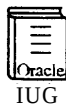
For more information: Appendix C “National Language Support” Also, see your installation or user's guide for linguistic sorts supported by your language and the sorting rules used by the linguistic sorting mechanisms.

NLS_TERRITORY

Default value: O/S dependent

Specifies the name of the territory whose conventions are to be followed for day and week numbering. Also specifies the default date format, the default decimal character and group separator, and the default ISO and local currency symbols. Supported territories include America, France, and Japan.

This parameter determines the default values for the following parameters: NLS_CURRENCY, NLS_ISO_CURRENCY, NLS_DATE_FORMAT, and NLS_NUMERIC_CHARACTERS.



For more information: Appendix C “National Language Support” See your installation or user's guide for the territory-dependent default values for these parameters.

OPEN_CURSORS

Default value: 50
Range of values: 1..O/S limit

The maximum number of open cursors (context areas) a single-user process can have at once. This parameter does not control a system-wide feature, but rather, the maximum address/memory space used by each process. The control of context areas is specific to the application.



For more information: Chapter 21 “Tuning Memory Allocation”. See your installation or user’s guide for the range of values.

OPEN_LINKS

Default value: 4
Range of values: 0..255

The maximum number of concurrent open connections to remote databases per user process. Value should equal or exceed the number of databases referred to in any single SQL statement that references multiple databases so that all the databases can be open in order to execute the statement. Value should be increased if many different databases are accessed over time. Thus, if queries alternately access databases A, B, and C and OPEN_LINKS is set to 2, time would be spent waiting while one connection was broken and another made.

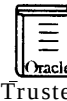
This parameter refers only to connections used for distributed transactions. Direct connections to a remote database specified as an application connects are not counted.

If set to 0, then no distributed transaction are allowed.

For more information: Chapter 15 “Managing Distributed Databases”

OPEN_MOUNTS

This is a Trusted ORACLE parameter.



For more information: *Trusted ORACLE7 Server Administrator’s Guide*

OPTIMIZER_MODE

Default value: COST
Range of values: RULE / COST

When set to RULE, this parameter causes rule-based optimization to be used, unless hints are specified in the query. When set to COST, the optimizer uses the cost-based approach for a SQL statement if there are statistics in the dictionary for at least one table accessed in the statement. (Otherwise, the rule-based approach is used.)

You can set the goal for cost based optimization within a session by using the ALTER SESSION command. This command lets you specify a goal of maximizing throughput or of minimizing response time.

For more information: *ORACLE7 Server Application Developer's Guide* Chapter 5 "Tuning SQL Statements" and the *ORACLE7 Server Concepts Manual* Chapter 13 "The Optimizer"

OS_AUTHENT_PREFIX

Default value: O/S dependent (typically "OP\$")

This parameter is used to authenticate users attempting to connect to the server with the users' operating system account name and password. The value of this parameter is concatenated to the beginning of every user's operating system account. The prefixed username is compared with the ORACLE usernames in the database when a connection request is attempted. The default value of this parameter is OP\$ for backward compatibility with previous versions. However, you might prefer to set the prefix value to "" (a null string) thereby eliminating the addition of any prefix to operating system account names.



Note: The text of the OS_AUTHENT_PREFIX parameter is case sensitive with some operating systems. Refer to your installation or user's guide for more information about this initialization parameter.

For more information: Chapter 10 "Establishing Security Policies"

OS_ROLES

Default value: false
Range of values: true/false

Setting this parameter to TRUE causes the database to allow the operating system to identify each username's roles. When a user attempts to create a session, the username's security domain is initialized using the roles identified by the operating system. A user can subsequently enable as many roles identified by the operating system as specified by the parameter MAX_OS_ROLES.

If OS_ROLES is equal to TRUE, the operating system completely manages the role grants for all database usernames. Any revokes of roles granted by the operating system are ignored, and any previously granted roles are ignored.

The default value, FALSE, causes roles to be identified and managed by the database.

For more information: Chapter 10 “Establishing Security Policies”

PROCESSES

Default value: 50
Range of values: 6 to O/S dependent
Multiple instances: can have different values

For a multiple-process operation, this parameter specifies the maximum number of operating system user processes that can simultaneously connect to an ORACLE server. Should include up to 6 for the background processes (or more if MI_BG_PROCS is non-zero or if you use the dispatcher configuration) plus one for logon; so a value of 20 would permit 13 or 14 concurrent users.



The default values of DDL_LOCKS and SESSIONS are derived from PROCESSES. If you alter the value of PROCESSES, you may want to adjust the values of these derived parameters.



For more information: Chapter 4 “Managing ORACLE Processes”. See your installation or user's guide for the range of values.

REMOTE_OS_AUTHENT

Default value: false
Range of values: true/false

Setting this parameter to TRUE allows authentication of remote clients with the value of OS_AUTHENT_PREFIX.

For more information: Chapter 15 “Managing Distributed Databases” and OS_AUTHENT_PREFIX on page A-38

REMOTE_OS_ROLES**Default value:** false**Range of values:** true/false

Setting this parameter to TRUE allows O/S roles for remote clients. The default value, FALSE, causes roles to be identified and managed by the database for remote clients.

For more information: Chapter 15 “Managing Distributed Databases” and OS_ROLES on page A-39

RESOURCE_LIMIT**Default value:** false**Range of values:** true/false

Changes the enforcement status of resource limits set in database profiles. A value of FALSE disables the enforcement of resource limits. A value of TRUE enables the enforcement of resource limits.

For more information: Chapter 11 “Controlling Access to the Database”

ROLLBACK_SEGMENTS

Default value: null (the instance uses public rollback segments by default if you do not specify this parameter)

Range of Values: Any rollback segment names listed in DBA_ROLLBACK_SEGS except SYSTEM

Multiple instances: must have different values (different instances cannot specify the same rollback segment)

One or more rollback segments to allocate by name to this instance. If ROLLBACK_SEGMENTS is not overridden, an instance acquires all of the rollback segments named in this parameter, even if the number of rollback segments exceeds the minimum number required by the instance (calculated from the ratio $\text{TRANSACTIONS} / \text{TRANSACTIONS\_PER\_ROLLBACK\_SEGMENT}$).

Note: Never name the SYSTEM rollback segment as a value for the ROLLBACK_SEGMENTS parameter.

This parameter has the following syntax:

ROLLBACK_SEGMENTS = (rbseg_name [, rbseg_name] ...)

Although this parameter usually specifies private rollback segments, it can also specify public rollback segments if they are not already in use.

Different instances in an ORACLE Parallel Server cannot name the same rollback segment for any of the ROLLBACK_SEGMENTS. Query the data dictionary view DBA_ROLLBACK_SEGS to find the name, segment ID number, and status of each rollback segment in the database.

For more information: Chapter 9 “Managing Rollback Segments”

ROW_CACHE_CURSORS

Default value: 10

Range of values: 10..3300

The number of cached recursive cursors used by the row cache manager for selecting rows from the data dictionary. The default value is sufficient for most systems.

For more information: *ORACLE7 Server Concepts Manual*, Chapter 9 “Memory Structures and Processes”

ROW_LOCKING

Default value: ALWAYS
Range of values: ALWAYS/INTENT
Multiple instances: must have the same value

The default of ALWAYS means that only row locks are acquired when a table is updated. INTENT means that only row locks are used on a SELECT FOR UPDATE, but at update time table locks are acquired.

For more information: *ORACLE7 Server Application Developer's Guide*, Chapter 5 "Tuning SQL Statements"

SEQUENCE_CACHE_ENTRIES

Default value: 10
Range of values: 10..32000
Multiple instances: can have different values

The number of sequences that can be cached in the SGA for immediate access. This cache is managed on a least recently used (LRU) basis, so if a request is made for a sequence that is not in the cache and there are no free entries, the oldest one on the LRU list is deleted and replaced with the newly requested one. Highest concurrency is achieved when this value is set to the highest possible number of sequences that will be used on an instance at one time.

Each entry requires approximately 110 bytes in the SGA for an ORACLE Parallel Server.

Sequences created with the NOCACHE option do not reside in this cache. They must be written through to the data dictionary on every use.

For more information: *ORACLE7 Server Application Developer's Guide*, Chapter 2 "Managing Schema Objects"

SEQUENCE_CACHE_HASH_BUCKETS

Default value: 7
Range of values: 1..32000 (if not a prime number, system uses next larger prime)
Multiple instances: can have different values

The number of buckets (at about 8 bytes per bucket) used to speed lookup for newly requested sequences in the cache. The cache is arranged as a hash table; a process making its first request for a sequence looks for it in this table.

This value should be prime; otherwise, the system uses the smallest prime number greater than or equal to the value you specify. Values larger than SEQUENCE_CACHE_ENTRIES are not meaningful.

For more information: *ORACLE7 Server Concepts Manual*, Chapter 9
“Memory Structures and Processes”

SERIALIZABLE

Default value: false
Range of values: true/false
Multiple instances: must have the same value

If TRUE, then queries acquire table-level read locks, preventing any update of objects read until the transaction containing the query is committed. This mode of operation provides repeatable reads and ensures that two queries for the same data within the same transaction see the same values.

Setting SERIALIZABLE to TRUE provides degree three consistency at a considerable cost in concurrency.

For more information: *ORACLE7 Server Concepts Manual*, Chapter 10
“Data Concurrency”

SESSIONS

Default value: derived (1.1 * PROCESSES)

The total number of user and system sessions. The default number is greater than PROCESSES to allow for recursive sessions.

The default value of DDL_LOCKS is derived from SESSIONS. If you alter the value of SESSIONS, you may want to adjust the value of DDL_LOCKS.

For more information: *ORACLE7 Server Concepts Manual*, Chapter 9
“Memory Structures and Processes”

SHARED_POOL_SIZE

Default value: 3.5 Mbytes
Range of values: 300 Kbytes - O/S dependent

The size of the shared pool, in bytes. The shared pool contains shared cursors and stored procedures. Larger values improve performance in multi-user systems. Smaller values use less memory.

For more information: Chapter 21 “Tuning Memory Allocation”

SINGLE_PROCESS

Default value: false
OK to change?: yes (only for an instance in exclusive mode with a single user)
Range of values: true/false
Multiple instances: must be false for all instances

Determines whether a database is brought up in single user (single process) mode or multiuser (multiprocess) mode. TRUE indicates single user and FALSE indicates multiuser. Value must be FALSE for an instance of an ORACLE Parallel Server.

For more information: Chapter 4 “Managing ORACLE Processes”

SMALL_TABLE_THRESHOLD

Default value: 4
Range of values: 0..O/S dependent

This parameter determines the number of buffers in the SGA that are available for scanning a table. Tables that are smaller than this value can be read into the cache in their entirety. When scanning a table that is larger than this value, buffers are reused immediately instead of aging in the cache. Except in very rare cases, the default setting provides optimal performance

For more information: *ORACLE7 Server Concepts Manual*, Chapter 9 “Memory Structures and Processes”

SORT_AREA_RETAINED_SIZE

Default value: derived (=SORT_AREA_SIZE)

Range of values: 0..SORT_AREA_SIZE

This is the maximum amount of session memory, in bytes, that will be used for an in-memory sort. This memory is released when the last row is fetched from the sort space.

If a sort requires more memory, a temporary segment is allocated and the sort becomes an external (disk) sort. The maximum amount of memory to use for the sort is then specified by SORT_AREA_SIZE (described below) instead of by this parameter.

Larger values permit more sorts to be performed in memory. However, multiple sort spaces of this size maybe allocated. Usually, only one or two sorts occur at one time, even for complex queries. In some cases, though, additional concurrent sorts are required. Each sort occurs in its own memory area, as specified by SORT_AREA_RETAINED_SIZE.

For more information: Chapter 24 “Additional Tuning Considerations”

SORT_AREA_SIZE

Default value: O/S dependent

The maximum amount of PGA memory to use for an external (disk) sort, specified in bytes. This memory is released when the sorted rows are written to disk.

Increasing SORT_AREA_SIZE size improves the efficiency of large sorts. Multiple allocations never exist there is only one memory area of SORT_AREA_SIZE for each user process at any given time.

The default is usually fine for most database operations. Only if very large indexes are being created might you want to adjust this parameter. For example, if one process is doing all database access, as in a full database import, then an increased value for this parameter may speed the import, particularly the CREATE INDEX statements.



For more information: Chapter 24 “Additional Tuning Considerations”. See your installation or user’s guide for the default value.

SORT_SPACEMAP_SIZE**Default value:** O/S dependent

The size in bytes of the sort spacemap in the context area. Only if you have very large indexes should you adjust this parameter. A sort automatically increases its spacemap if necessary, but it does not necessarily do so when it will make best use of disk storage. The sort makes optimal use of disk storage if SORT_SPACEMAP_SIZE is set to

$$[(\text{total-sort-bytes}) / (\text{sort-area-size})] + 64$$

where *total-sort-bytes* is:

$$(\text{number-of-records}) * [\text{sum-of-average-column-sizes} + (2 * \text{number of col})]$$

where columns include the SELECT list for the ORDER BY, the SELECT list for the GROUP BY, and the key list for CREATE INDEX. Also include 10 bytes for ROWID for CREATE INDEX and GROUP BY or ORDER BY columns not mentioned in the SELECT list for these cases.



For more information: *ORACLE7 Server Concepts Manual*, Chapter 9 “Memory Structures and Processes”. See your installation or user’s guide for the default value.

SQL_TRACE**Default value:** false**Range of values:** true/false

Disables or enables the SQL trace facility. Setting this parameter to TRUE provides information on tuning that you can use to improve performance. Because the SQL trace facility causes system overhead, you should run the database with the value TRUE only for the purposes of collecting statistics.

The ALTER SESSION command can override the setting of the SQL_TRACE parameter during a particular session.

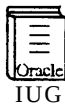
For more information: *ORACLE7 Server Application Developer’s Guide*, Appendix B “Performance Diagnostic Tools” and the *ORACLE7 Server SQL Language Reference Manual*

TEMPORARY_TABLE_LOCKS

Default value: derived (=SESSIONS)

Range of values: 0..O/S dependent

Determines the number of temporary tables that can be created in the temporary segment space. A temporary table lock is needed any time a sort occurs that is too large to hold in memory, either as the result of a select on a large table with ORDER BY or as a result of sorting a large index. Installations with many users of applications that simultaneously perform several ordered queries on large tables may need to increase this number. Most installations should do well with the default.



For more information: Chapter 24 “Additional Tuning Considerations”. See your installation or user’s guide for the range of values.

THREAD

Default value: 0

Range of values: 0..maximum number of declared threads

Multiple instances: if specified, must have different values

This parameter is applicable only to instances that intend to run in parallel (shared) mode.

The number of the redo thread that is to be used by the instance. Any available redo thread number can be used, but an instance cannot use the same thread number as another instance. Also, an instance cannot start when its redo thread is disabled. A value of zero causes an available, enabled public thread to be chosen. An instance cannot mount a database if the thread is used by another instance or if the thread is disabled.

Redo threads are specified with the THREAD option of the ALTER DATABASE ADD LOGFILE command. Redo threads are enabled with the ALTER DATABASE ENABLE [PUBLIC] THREAD command. The PUBLIC keyword signifies that the redo thread may be used by any instance.

Thread 1 is the default thread in exclusive mode. An instance running in exclusive mode can specify THREAD to use the redo log files in a thread other than thread 1.

For more information: *ORACLE7 Parallel Server Administrator’s Guide* and the *ORACLE7 Server SQL Language Reference Manual*

TIMED_STATISTICS

Default value: false
Range of values: true/false

By default (when set to FALSE), the SQL*DBA statistics related to time (from the buffer manager) always are zero and the Server can avoid the overhead of requesting the time from the operating system. To turn on statistics, set the value to TRUE. Should normally be set to FALSE.

For more information: *ORACLE7 Server Application Developer's Guide*, Appendix B "Performance Diagnostic Tools"

TRANSACTIONS

Default value: derived (1.1 * PROCESSES)
Multiple instances: can have different values

The maximum number of concurrent transactions. Greater values increase the size of the SGA and can increase the number of rollback segments allocated. The default value is greater than PROCESSES to allow for recursive transactions.

For more information: *ORACLE7 Server Concepts Manual*, Chapter 9 "Memory Structures and Processes" and Chapter 12 "Transaction Management"

TRANSACTIONS_PER_ROLLBACK_SEGMENT

Default value: 30
Range of values: 1..O/S dependent
Multiple instances: can have different values

The number of concurrent transactions allowed per rollback segment. The minimum number of rollback segments acquired at startup is TRANSACTIONS divided by the value for this parameter. For example, if TRANSACTIONS is 101 and this parameter is 10, then the minimum number of rollback segments acquired would be the ratio 101/10, rounded up to 11.

More rollback segments can be acquired if they are identified with ROLLBACK_SEGMENT_INITIAL and ROLLBACK_SEGMENT_COUNT or named in ROLLBACK_SEGMENTS.



For more information: Chapter 9 "Managing Rollback Segments". See your installation or user's guide for the range of values.

USER_DUMP_DEST

Default value: O/S dependent

Range of values: valid local pathname, directory, or disk

The path name for a directory where debugging trace files from a user process are written.



For more information: *ORACLE7 Server Application Developer's Guide*, Appendix B "Performance Diagnostic Tools". See your installation or user's guide for the range of values.

B

DATA DICTIONARY REFERENCE

This appendix contains descriptions of the data dictionary tables and views. To see the current data dictionary on your system, query the view `DICTIONARY`.

This appendix contains the following information:

- SQL scripts available for altering the data dictionary or granting access to tables (page B-2)
- Data dictionary views and their columns (page B-5)
- Views of the dynamic performance tables—the V\$ views (page B-47)



Trusted

In Trusted ORACLE, each of the dictionary tables and views contains the column which indicates the label of each row in the table or view. There are also additional dictionary tables and views in Trusted ORACLE, as well as additional columns in some tables and views. See the *Trusted ORACLE7 Server Administrator's Guide* for more information.

Data Dictionary SQL Script Files



Certain ORACLE SQL script files perform data dictionary functions and can be run during or after database creation. The following is a list of ORACLE SQL scripts that modify the data dictionary. The standard script names are shown. On some operating systems, the names may differ. Consult your installation or user's guide for details.

Standard Scripts

These scripts are run by all installations.

SQL.BSQ	Database bootstrap script.
CATALOG.SQL	Creates the standard data dictionary views.

Data Dictionary Scripts

These scripts are run by SYS to create data dictionary views.

CATAUDIT.SQL	Creates the auditing data dictionary views.
CATBLOCK.SQL	Creates a view called BLOCKING_LOCKS. You can query this view to find out which locks are blocking the system; for example, you could find processes that have died while holding locks, which for some reason have not been cleaned up by PMON.
CATDBSYN.SQL	Creates private synonyms for the DBA data dictionary views.
CATEXP.SQL	Creates roles for import and export and incremental export views.
CATLDR.SQL	Creates SQL*Loader views for direct loads.
CATNOAUD.SQL	Drops the auditing tables.
CATNOPRC.SQL	Drops the procedural object views.
CATPARR.SQL	Creates data dictionary views used for installations with the Parallel Server option.
CATPROC.SQL	Creates data dictionary views for procedural objects and snapshots used in installations with the procedural option.
CATTRUST.SQL	Creates views for Trusted ORACLE.
CATSTAT.SQL	Computes statistics for tables and views created by SQL.BSQ for cost-based optimization.
CATSNAP.SQL	Creates views for storing and maintaining snapshots.

CATRPC.SQL	Creates data dictionary views on distributed database information
STANDARD.SQL	Creates PL/SQL packages for the procedural option
DBMSSTDY.SQL	Includes extensions to the package standard

Procedure Scripts

These scripts are also run by SYS. They create stored procedures and packages.

DBMSALRT.SQL	Creates the user-alert procedures and one supporting view.
DBMSLOCK.SQL	Creates PL/SQL lock management packages that allow applications to create locks, give them names, request them, and release them, as well as to convert a lock from one of several supported lock modes to another. Routines are also provided to wait for a period of time before re-requesting a lock. Deadlock conditions among the locks are detected.
DBMSMAIL.SQL	Creates PL/SQL package to send mail. This routine can be called from a trigger. See also: UTLMAIL.SQL.
DBMSOTPT.SQL	Creates output procedures for use in script debugging and information display.
DBMSPIPE.SQL	Creates procedures for inter-process communication.
DBMSSNAP.SQL	Creates snapshot refresh procedures.
DBMSUTIL.SQL	Creates PL/SQL utility packages, including routines for transaction management, session control, and DDL statement execution. Also supports recompilation of all stored PL/SQL objects.
DBMSDESC.SQL	

User Scripts

These scripts are run by users.

UTLBSTAT.SQL	Creates statistics tables and collects beginning statistics.
UTLCHAIN.SQL	Creates the default table for storing the output of the ANALYZE LIST CHAINED ROWS command.
UTLDIDX.SQL	Displays index statistics captured by UTLIDXSS.SQL and UTLIDXSO.SQL.
UTLDTREE.SQL	Creates the procedure and view used to show all objects (to which a user has access) that are dependent on a given object, including recursive dependency.
UTLESTAT.SQL	Collects ending statistics and generates a report.
UTLEXCPT.SQL	Creates the exceptions table for recording violations of integrity constraints.
UTLLOCKT.SQL	Shows the lock wait-for graph in tree-structured form to help diagnose systems that are hung on locks.
UTLMAIL.SQL	Upgrades the mail database so that it can be accessed via Remote Procedure Calls (RPC). See also: DBMSMAIL.SQL.
UTLMONTR.SQL	Grants access to the dynamic performance tables to PUBLIC.
UTLOIDX.SQL	Analyzes a column of a table, generating statistics that can be used to evaluate its suitability for indexing.
UTLSAMPL.SQL	Creates the initial database users and tables for experimentation and illustration.
UTLSIDX.SQL	Analyzes index statistics for existing index columns. Can analyze multiple tables/columns at one time.
UTLXPLAN.SQL	Creates the table used by EXPLAIN PLAN to store the description of the statement execution plan.

Migration Scripts

For details on these scripts, see the *ORACLE7 Server Migration Guide*.

CATALOG6.SQL	Creates Version 6 views for compatibility.
CATEXP6.SQL	Creates the views needed for a Version 6 export.
DROPCAT6.SQL	Removes the Version 6 views and restores the data dictionary to full ORACLE7 form.

Data Dictionary Views

The following is an alphabetical reference of the data dictionary views accessible to all users of an ORACLE server. Most views can be accessed by any user with the CREATE_SESSION privilege.

The data dictionary views that begin with DBA_ are restricted. These views can be accessed only by users with the SELECT_ANY_TABLE privilege. This privilege is assigned to the DBA role when the system is initially installed.

ALL_CATALOG

All tables, views, synonyms, and sequences accessible to the user

OWNER	Owner of the object
TABLE_NAME	Name of the object
TABLE_TYPE	Type of the object

ALL_COL_COMMENTS

Comments on columns of accessible tables and views

OWNER	Owner of the object
TABLE_NAME	Name of the object
COLUMN_NAME	Name of the column
COMMENTS	Comment on the column

ALL_COL_PRIVS

Grants on columns for which the user or PUBLIC is the grantee

GRANTOR	Name of the user who performed the grant
GRANTEE	Name of the user to whom access was granted
TABLE_SCHEMA	Schema of the object
TABLE_NAME	Name of the object
COLUMN_NAME	Name of the column
PRIVILEGE	Privilege on the column
GRANTABLE	YES if the privileges was granted with ADMIN OPTION, otherwise NO

ALL_COL_PRIVS_MADE

Grants on columns for which the user is owner or grantor

GRANTEE	Name of the user to whom access was granted
OWNER	Username of the owner of the object
TABLE_NAME	Name of the object
COLUMN_NAME	Name of the column
GRANTOR	Name of the user who performed the grant
PRIVILEGE	Privilege on the column

	GRANTABLE	YES if the privilege was granted with ADMIN OPTION, otherwise NO
ALL_COL_PRIVS_RECD	Grants on columns for which the user or PUBLIC is the grantee	
	GRANTEE	Name of the user to whom access was granted
	OWNER	Username of the owner of the object
	TABLE_NAME	Name of the object
	COLUMN_NAME	Name of the column
	GRANTOR	Name of the user who performed the grant
	PRIVILEGE	Privilege on the column
	GRANTABLE	YES if the privilege was granted with ADMIN OPTION, otherwise NO
ALL_CONSTRAINTS	Constraint definitions on accessible tables	
	OWNER	Owner of the constraint definition
	CONSTRAINT_NAME	Name associated with the constraint definition
	CONSTRAINT_TYPE	Type of constraint definition: C (check constraint on a table), P (primary key), U (unique key), R (referential integrity), or V (with check option, on a view)
	TABLE_NAME	Name associated with table with constraint definition
	SEARCH_CONDITION	Text of search condition for table check
	R_OWNER	Owner of table used in referential constraint
	R_CONSTRAINT_NAME	Name of unique constraint definition for referenced table
	DELETE_RULE	Delete rule for a referential constraint: CASCADE / NO ACTION
	STATUS	Status of constraint ENABLED or DISABLED
ALL_CONS_COLUMNS	Information about accessible columns in constraint definitions	
	OWNER	Owner of the constraint definition
	CONSTRAINT_NAME	Name associated with the constraint definition
	TABLE_NAME	Name associated with table with constraint definition
	COLUMN_NAME	Name associated with column specified in the constraint definition
	POSITION	Original position of column in definition

ALL_DB_LINKS	Database links accessible to the user	
	OWNER	Username of the owner of the database link
	DB_LINK	Name of the database link
	USERNAME	Name of user when logging on
	HOST	SQL*Net string for connect
	CREATED	Creation time of the database link
ALL_DEF_AUDIT_OPTS	Default object-auditing options that will be applied when objects are created	
	ALT	Auditing ALTER WHENEVER SUCCESSFUL / UNSUCCESSFUL
	AUD	Auditing AUDIT WHENEVER SUCCESSFUL / UNSUCCESSFUL
	COM	Auditing COMMENT WHENEVER SUCCESSFUL / UNSUCCESSFUL
	DEL	Auditing DELETE WHENEVER SUCCESSFUL / UNSUCCESSFUL
	GRA	Auditing GRANT WHENEVER SUCCESSFUL / UNSUCCESSFUL
	IND	Auditing INDEX WHENEVER SUCCESSFUL / UNSUCCESSFUL
	INS	Auditing INSERT WHENEVER SUCCESSFUL / UNSUCCESSFUL
	LOC	Auditing LOCK WHENEVER SUCCESSFUL / UNSUCCESSFUL
	REN	Auditing RENAME WHENEVER SUCCESSFUL / UNSUCCESSFUL
	SEL	Auditing SELECT WHENEVER SUCCESSFUL / UNSUCCESSFUL
	UPD	Auditing UPDATE WHENEVER SUCCESSFUL / UNSUCCESSFUL
	REF	Auditing REFERENCES WHENEVER SUCCESSFUL / UNSUCCESSFUL
	EXE	Auditing EXECUTE WHENEVER SUCCESSFUL / UNSUCCESSFUL

ALL_DEPENDENCIES**Dependencies between objects accessible to the user**

OWNER	Owner of the object
NAME	Name of the object
TYPE	Type of object: PROCEDURE, PACKAGE, FUNCTION, PACKAGE BODY
REFERENCED_OWNER	Owner of the parent object
REFERENCED_NAME	Name of the parent object
REFERENCED_TYPE	Type of the parent object PROCEDURE, PACKAGE, FUNCTION, PACKAGE BODY
REFERENCED_LINK_NAME	Name of the link to the parent object (if remote)

ALL_ERRORS**Current errors on all objects accessible to the user**

OWNER	Owner of the object
NAME	Name of the object
TYPE	Type of object VIEW, PROCEDURE, PACKAGE, FUNCTION, PACKAGE BODY
SEQUENCE	Sequence number, for ordering
LINE	Line number at which this error occurs
POSITION	Position in the line at which this error occurs
TEXT	Text of the error

ALL_INDEXES**Description of indexes on tables accessible to the user**

OWNER	Username of the owner of the index
STATUS	State of the index: DIRECT LOAD or VALID
INDEX_NAME	Name of the index
TABLE_OWNER	Owner of the indexed object
TABLE_NAME	Name of the indexed object
TABLE_TYPE	Type of the indexed object
UNIQUENESS	Uniqueness status of the index: UNIQUE or NONUNIQUE
TABLESPACE_NAME	Name of the tablespace containing the index
INI_TRANS	Initial number of transactions
MAX_TRANS	Maximum number of transactions
INITIAL_EXTENT	Size of the initial extent
NEXT_EXTENT	Size of secondary extents
MIN_EXTENTS	Minimum number of extents allowed in the segment
MAX_EXTENTS	Maximum number of extents allowed in the segment
PCT_INCREASE	Percentage increase in extent size
PCT_FREE	Minimum percentage of free space in a block
BLEVEL	Depth of the index from its root block to its leaf blocks. A depth of 1 indicates that the root block and leaf block are the same.
LEAF_BLOCKS	Number of leaf blocks in the index.
DISTINCT_KEYS	Number of distinct indexed values. For indexes that enforce UNIQUE and PRIMARY KEY constraints, this value is the same as the number of rows in the table (USER_TABLES.NUM_ROWS).
AVG_LEAF_BLOCKS_ PER_KEY	Average number of leaf blocks in which each distinct value in the index appears. This statistic is rounded to the nearest integer. For indexes that enforce UNIQUE and PRIMARY KEY constraints, this value is always 1.
AVG_DATA_BLOCKS_ PER_KEY	Average number of data blocks in the table that are pointed to by a distinct value in the index. This statistic is the average

number of data blocks that contain rows that contain a given value for the indexed columns. This statistic is rounded to the nearest integer.

CLUSTERING_FACTOR

Statistic that represents the amount of order of the rows in the table based on the values of the index.

If its value is near the number of blocks, then the table is very well ordered. In such a case, the index entries in a single leaf block tend to point to rows in the same data blocks.

If its value is near the number of rows, then the table is very randomly ordered. In such a case, it is unlikely that index entries in the same leaf block point to rows in the same data blocks.

ALL_IND_COLUMNS

Columns of the indexes on accessible tables

INDEX_OWNER	Index owner
INDEX_NAME	index name
TABLE_OWNER	Table or cluster owner
TABLE_NAME	Table or cluster name
COLUMN_NAME	Column name
COLUMN_POSITION	Position of column within index
COLUMN_LENGTH	Indexed length of the column

ALL_LABELS



Trusted

A TRUSTED ORACLE view that lists system labels. For details, see the *Trusted ORACLE7 Server Administrator's Guide*.

ALL_MOUNTED_DBS



Trusted

A TRUSTED ORACLE view that lists mounted databases. For details, see the *Trusted ORACLE7 Server Administrator's Guide*.

ALL_OBJECTS**Objects accessible to the user**

OWNER	Username of the owner of the object
OBJECT_NAME	Name of the object
OBJECT_ID	Object number of the object
OBJECT_TYPE	Type of the object
CREATED	Timestamp for the creation of the object
LAST_DDL_TIME	Timestamp for the last modification of the object resulting from a DDL command (including grants and revokes)
TIMESTAMP	Timestamp for the creation of the object (character data)
STATUS	Status of the object VALID, INVALID, or N / A

ALL_SEQUENCES**Description of sequences accessible to the user**

SEQUENCE_OWNER	Name of the owner of the sequence
SEQUENCE_NAME	Sequence name
MIN_VALUE	Minimum value of the sequence
MAX_VALUE	Maximum value of the sequence
INCREMENT_BY	Value by which sequence is incremented
CYCLE_FLAG	Does sequence wrap around on reaching limit
ORDER_FLAG	Are sequence numbers generated in order
CACHE_SIZE	Number of sequence numbers to cache
LAST_NUMBER	Last sequence number written to disk. If a sequence uses caching, the number written to disk is the last number placed in the sequence cache. This number is likely to be greater than the last sequence number that was used.

ALL_SNAPSHOTS

All snapshots accessible to the user

OWNER	Owner of the snapshot
NAME	Name of the view used by users and applications for viewing the snapshot
TABLE_NAME	Table the snapshot is stored in. This table has an extra column for the master rowid
MASTER_VIEW	View of the master table, owned by the snapshot owner, used for refreshes
MASTER_OWNER	Owner of the master table
MASTER	Name of the master table that this snapshot is a copy of
MASTER_LINK	Database link name to the master site
CAN_USE_LOG	YES if this snapshot can use a snapshot log, NO if this snapshot is too complex to use a log
LAST_REFRESH	Date and time at the master site of the last refresh
ERROR	Error returned last time an automatic refresh was attempted
TYPE	Type of refresh for all automatic refreshes: COMPLETE, FAST, FORCE
NEXT	Date function used to compute next refresh dates
START_WITH	Date function used to compute next refresh dates
QUERY	Original query that this snapshot is an instantiation of

ALL_SOURCE**Text source of all stored objects accessible to the user**

OWNER	Owner of the object
NAME	Name of the object
TYPE	Type of object: PROCEDURE, PACKAGE, FUNCTION, PACKAGE BODY
LINE	Line number of this line of source
TEXT	Text source of the stored object

ALL_SYNONYMS**All synonyms accessible to the user**

OWNER	Owner of the synonym
SYNONYM_NAME	Name of the synonym
TABLE_OWNER	Owner of the object referenced by the synonym
TABLE_NAME	Name of the object referenced by the synonym
DB_LINK	Name of the database link referenced, if any

ALL_TABLES**Description of tables accessible to the user**

OWNER	Owner of the table
TABLE_NAME	Name of the table
TABLESPACE_NAME	Name of the tablespace containing the table
CLUSTER_NAME	Name of the cluster, if any, to which the table belongs
PCT_FREE	Minimum percentage of free space in a block
PCT_USED	Minimum percentage of used space in a block
INI_TRANS	Initial number of transactions
MAX_TRANS	Maximum number of transactions
INITIAL_EXTENT	Size of the initial extent in bytes
NEXT_EXTENT	Size of secondary extents in bytes
MIN_EXTENTS	Minimum number of extents allowed in the segment
MAX_EXTENTS	Maximum number of extents allowed in the segment
PCT_INCREASE	Percentage increase in extent size
BACKED_UP	Has table been backed up since last change
NUM_ROWS	Number of rows in the table
BLOCKS	Number of data blocks allocated to the table
EMPTY_BLOCKS	Number of data blocks allocated to the table that contain no data
AVG_SPACE	Average amount of free space (in bytes) in a data block allocated to the table
CHAIN_CNT	Number of rows in the table that are chained from one data block to another, or which have migrated to a new block, requiring a link to preserve the old rowid
AVG_ROW_LEN	Average length of a row in the table in bytes

ALL_TAB_COLUMNS	Columns of all tables, views, and clusters accessible to the user	
OWNER	OWNER	Owner of the table, view or cluster
TABLE_NAME	TABLE_NAME	Table, view, or cluster name
COLUMN_NAME	COLUMN_NAME	Column name
DATA_TYPE	DATA_TYPE	Datatype of the column
DATA_LENGTH	DATA_LENGTH	Length of the column in bytes
DATA_PRECISION	DATA_PRECISION	Decimal precision for NUMBER datatype; binary precision for FLOAT datatype; NULL for all other datatypes.
DATA_SCALE	DATA_SCALE	Digits to right of decimal point in a number
NULLABLE	NULLABLE	Does column allow NULLS? Value is N if there is a NOT NULL constraint on the column or if the column is part of a PRIMARY KEY.
COLUMN_ID	COLUMN_ID	Sequence number of the column as created
DEFAULT_LENGTH	DEFAULT_LENGTH	Length of default value for the column
DATA_DEFAULT	DATA_DEFAULT	Default value for the column
NUM_DISTINCT	NUM_DISTINCT	Number of distinct values in each column of the table
LOW_VALUE	LOW_VALUE	For tables with more than three rows, the second-lowest and second-highest values in the column. For tables with three rows or fewer, the lowest and highest values. These statistics are expressed in hexadecimal notation for the internal representation of the first 32 bytes of the values.
HIGH_VALUE	HIGH_VALUE	
ALL_TAB_COMMENTS	Comments on tables and views accessible to the user	
OWNER	OWNER	Owner of the object
TABLE_NAME	TABLE_NAME	Name of the object
TABLE_TYPE	TABLE_TYPE	Type of the object
COMMENTS	COMMENTS	Comment on the object

ALL_TAB_PRIVS**Grants on objects for which the user or PUBLIC is the grantee**

GRANTOR	Name of the user who performed the grant
GRANTEE	Name of the user to whom access is granted
TABLE_SCHEMA	Schema of the object
TABLE_NAME	Name of the object
PRIVILEGE	Privilege on the object
GRANTABLE	YES if the privilege was granted with ADMIN OPTION, otherwise NO

ALL_TAB_PRIVS_MADE**User's grants and grants on user's objects**

GRANTEE	Name of the user to whom access was granted
OWNER	Owner of the object
TABLE_NAME	Name of the object
GRANTOR	Name of the user who performed the grant
PRIVILEGE	Privilege on the object
GRANTABLE	YES if the privilege was granted with ADMIN OPTION, otherwise NO

ALL_TAB_PRIVS_RECD**Grants on objects for which the user or PUBLIC is the grantee**

GRANTEE	Name of the user to whom access was granted
OWNER	Owner of the object
TABLE_NAME	Name of the object
GRANTOR	Name of the user who performed the grant
PRIVILEGE	Privilege on the object
GRANTABLE	YES if the privilege was granted with ADMIN OPTION, otherwise NO

ALL_TRIGGERS**Triggers accessible to the user**

OWNER	Owner of the trigger
TRIGGER_NAME	Name of the trigger
TRIGGER_TYPE	Type of the trigger: BEFORE ROW, BEFORE STATEMENT, AFTER ROW, AFTER STATEMENT
TRIGGERING_EVENT	Statement that fires the trigger: INSERT, UPDATE, DELETE
TABLE_OWNER	Owner of the table the trigger is defined on
TABLE_NAME	Table the trigger is defined on
REFERENCING_NAMES	Names used for referencing OLD and NEW column values from within the trigger
WHEN_CLAUSE	WHEN clause. Must evaluate to TRUE for TRIGGER_BODY to execute
STATUS	Whether or not the trigger is enabled: ENABLED or DISABLED
DESCRIPTION	Trigger description. Useful for re-creating a trigger creation statement.
TRIGGER_BODY	Statement(s) executed by the trigger when it fires

ALL_TRIGGER_COLS**Shows usage of columns in triggers owned by user, on tables owned by user, or on all triggers if the user has tech CREATE ANY TRIGGER privilege.**

TRIGGER_OWNER	Owner of the trigger
TRIGGER_NAME	Name of the trigger
TABLE_OWNER	Owner of the table the trigger is defined on
TABLE_NAME	Table the trigger is defined on
COLUMN_NAME	Name of the column used in the trigger
COLUMN_LIST	Column specified in UPDATE clause: Y / N
COLUMN_USAGE	How the column is used in the trigger. All applicable combinations of NEW, OLD, IN, OUT, and IN OUT.

ALL_USERS**Information about all users of the database**

USERNAME	Name of the user
USER_ID	ID number of the user
CREATED	User creation date

ALL_VIEWS	Text of views accessible to the user	
	OWNER	Owner of the view
	VIEW_NAME	Name of the view
	TEXT_LENGTH	Length of the view text
	TEXT	View text
AUDIT_ACTIONS	Description table for audit trail action type codes	
	ACTION	Numeric audit trail action type code
	NAME	Name of the type of audit trail action
CAT	Synonym for USER_CATALOG	
CHAINED_ROWS	Default table for the ANALYZE LIST CHAINED ROWS command.	
	OWNER_NAME	Table owner
	TABLE_NAME	Table name
	CLUSTER_NAME	Cluster the table is in, if any
	HEAD_ROWID	RowID the chained row is accessed by
	TIMESTAMP	Date/time that the ANALYZE command was issued
CLU	Synonym for USER_CLUSTERS	
CODE_PIECES	Accessed to create the ALL_OBJECT_SIZE, DBA_OBJECT_SIZE, and USER_OBJECT_SIZE views	
CODE_SIZE	Accessed to create the ALL_OBJECT_SIZE, DBA_OBJECT_SIZE, and USER_OBJECT_SIZE views	
COLS	Synonym for USER_TAB_COLUMNS	
COLUMN_PRIVILEGES	Grants on columns for which the user is the grantor, grantee, or owner, or PUBLIC is the grantee	
	GRANTEE	Name of the user to whom access was granted
	OWNER	Username of the object's owner
	TABLE_NAME	Name of the object
	COLUMN_NAME	Name of the column
	GRANTOR	Name of the user who performed the grant
	INSERT_PRIV	Permission to insert into the column
	UPDATE_PRIV	Permission to update the column
	REFERENCES_PRIV	Permission to reference the column
	CREATED	Timestamp for the grant

DBA_2PC_NEIGHBORS**Information about incoming and outgoing connections for pending transactions**

LOCAL_TRAN_ID	Transaction ID on the local database in the format: <i>n.n.n</i>
IN_OUT	IN for incoming transactions, OUT for outgoing transactions
DATABASE	For incoming transactions, the name of the client database; for outgoing transactions, the name of the database link
DBUSER_OWNER	For incoming transactions, the name of the local user; for outgoing transactions, the owner of the database link
INTERFACE	C signifies a request for commit; N signifies prepared state or a request for a read only commit.
DBID	Remote database ID
SESS#	Local session number
BRANCH	Local transaction branch ID

DBA_2PC_PENDING**Information about failed distributed transactions in PREPARED state**

LOCAL_TRAN_ID	Transaction ID on the local database in the format <i>n.n.n</i>
GLOBAL_TRAN_ID	Global transaction ID in the format <i>dbname.yy-mm-dd.hh:mi:ss.cs.n.n.n</i>
STATE	State of the transaction: COLLECTING, PREPARED, COMMITTED, HEURISTIC COMMIT, HEURISTIC ABORT
MIXED	Value of this column is YES if heuristic damage (mixed states) is detected
ADVICE	Action suggested to take: C (commit), R (rollback), NULL (no advice)
TRAN_COMMENT	Comment given with the COMMIT WORK COMMENT statement
FAIL_TIME	Date and time when the row was inserted
FORCE_TIME	Time of a manual force decision (NULL if the transaction was not forced locally)
RETRY_TIME	Time that automatic Recoverer process last tried to recover the transaction
OS_USER	Operating system account for the user
OS_TERMINAL	Operating system terminal ID for the user
HOST	Name of the host
DB_USER	ORACLE username of the user at the commit point site
COMMIT#	Global commit number for committed transactions

DBA_AUDIT_EXISTS**Audit trail entries created by the AUDIT EXISTS command**

OS_USERNAME	Operating system logon username of the user whose actions were audited
USERNAME	Name (not ID number) of the user whose actions were audited
USERHOST	Numeric instance ID for the ORACLE instance from which the user is accessing the database
TERMINAL	Identifier for the user's terminal
TIMESTAMP	Timestamp for the creation of the audit trail entry
OWNER	Intended creator of the non-existent object
OBJ_NAME	Name of the object affected by the action
ACTION_NAME	Name of the action type corresponding to the numeric code in ACTION
NEW_OWNER	Owner of the object named in the NEW_NAME column
NEW_NAME	New name of an object after a RENAME or the name of the underlying object
OBJ_PRIVILEGE	Object privileges granted / revoked by a GRANT / REVOKE statement
SYS_PRIVILEGE	System privileges granted / revoked by a GRANT / REVOKE statement
GRANTEE	Name of the grantee specified in a GRANT / REVOKE statement
SESSIONID	Numeric ID for each ORACLE session
ENTRYID	Numeric ID for each audit trail entry in the session
STATEMENTID	Numeric ID for each statement run (a statement may cause many actions)
RETURNCODE	ORACLE message code generated by the action. Some useful values follow: zero the action succeeded 2004 security violation

DBA_AUDIT_OBJECT	All audit trail records for all objects in the system	
	OS_USERNAME	Operating system logon user name of the user whose actions were audited
	USERNAME	Name (not ID number) of the user whose actions were audited
	USERHOST	Numeric instance ID for the ORACLE instance from which the user is accessing the database
	TERMINAL	Identifier for the user's terminal
	TIMESTAMP	Timestamp for the creation of the audit trail entry or logon time for the CONNECT statement
	OWNER	Creator of object affected by the action
	OBJ_NAME	Name of the object affected by the action
	ACTION_NAME	Name of the action type corresponding to the numeric code in ACTION
	NEW_OWNER	Owner of the object named in the NEW_NAME column
	NEW_NAME	New name of an object after a RENAME or the name of the underlying object
	SES_ACTIONS	Session summary (a string of 11 characters, one for each action type, in the order Alter, Audit, Comment, Delete, Grant, Index, Insert, Lock, Rename, Select, and Update; coded - for none, S for success, F for failure, and B for both)
	COMMENT_TEXT	Text comment on the audit trail entry (inserted by art application program)
	SESSIONID	Numeric ID for each ORACLE session
	ENTRYID	Numeric ID for each audit trail entry in the session
	STATEMENTID	Numeric ID for each statement run (a statement may cause many actions)
	RETURNCODE	ORACLE message code generated by the action (zero if the action succeeded)
	PRIV_USED	System privilege used to execute the action
	OBJECT_LABEL	Optional Trusted ORACLE label associated with the object being audited
	SESSION_LABEL	Trusted ORACLE label associated with the user session

DBA_AUDIT_SESSION**All audit trail records in the system concerning CONNECT and DISCONNECT**

OS_USERNAME	Operating system logon user name of the user whose actions were audited
USERNAME	Name (not ID number) of the user whose actions were audited
USERHOST	Numeric instance ID for the ORACLE instance from which the user is accessing the database
TERMINAL	Identifier for the user's terminal
TIMESTAMP	Timestamp for the creation of the audit trail entry or logon time for the CONNECT statement
ACTION_NAME	Name of the action type corresponding to the numeric code in ACTION
LOGOFF_TIME	Timestamp for user logoff
LOGOFF_LREAD	Logical reads for the session
LOGOFF_PREAD	Physical reads for the session
LOGOFF_LWRITE	Logical writes for the session
LOGOFF_DLOCK	Deadlocks detected during the session
SESSIONID	Numeric ID for each ORACLE session
RETURNCODE	ORACLE message code generated by the action (zero if the action succeeded)
SESSION_LABEL	Trusted ORACLE label associated with the user session

DBA_AUDIT_STATEMENT

All audit trail records in the system concerning GRANT, REVOKE, AUDIT, NOAUDIT, and ALTER SYSTEM statements

OS_USERNAME	Operating system logon user name of the user whose actions were audited
USERNAME	Name (not ID number) of the user whose actions were audited
USERHOST	Numeric instance ID for the ORACLE instance from which the user is accessing the database
TERMINAL	Identifier for the user's terminal
TIMESTAMP	Timestamp for the creation of the audit trail entry or logon time for the CONNECT statement
OWNER	Creator of object affected by the action
OBJ_NAME	Name of the object affected by the action
ACTION_NAME	Name of the action corresponding to the numeric code in ACTION
NEW_NAME	New name of an object renamed by a RENAME statement
OBJ_PRIVILEGE	Object privileges granted/revoked by a GRANT/REVOKE statement
SYS_PRIVILEGE	System privileges granted/revoked by a GRANT/REVOKE statement
ADMIN_OPTION	Signifies the role or system privilege was granted with ADMIN option
GRANTEE	Username of the grantee specified in a GRANT/REVOKE statement
AUDIT_OPTION	Auditing option set with the AUDIT statement
SES_ACTIONS	Session summary (a string of 11 characters, one for each action type, in the order Alter, Audit, Comment, Delete, Grant, Index, Insert, Lock, Rename, Select, and Update; coded - for none, S for success, F for failure, and B for both)
COMMENT_TEXT	Text comment on the audit trail entry (inserted by an application program)
SESSIONID	Numeric ID for each ORACLE session
ENTRYID	Numeric ID for each audit trail entry in the session
STATEMENTID	Numeric ID for each statement run (a statement may cause many actions)

RETURNCODE	ORACLE message code generated by the action (zero if the action succeeded)
PRIV_USED	System privilege used to execute the action
SESSION_LABEL	Trusted ORACLE label associated with the user session

DBA_AUDIT_TRAIL

Collection of all audit records in the system

OS_USERNAME	Operating system logon user name of the user whose actions were audited
USERNAME	Name (not ID number) of the user whose actions were audited
USERHOST	Numeric instance ID for the ORACLE instance from which the user is accessing the database
TERMINAL	Identifier for the user's terminal
TIMESTAMP	Timestamp for the creation of the audit trail entry or logon time for the CONNECT statement
OWNER	Creator of object affected by the action
OBJ_NAME	Name of the object affected by the action
ACTION	Numeric type code corresponding to the action name
ACTION_NAME	Name of the action type corresponding to the numeric code in ACTION
NEW_OWNER	Owner of the object named in the NEW_NAME column
NEW_NAME	New name of an object after a RENAME or the name of the underlying object
OBJ_PRIVILEGE	Object privileges granted /revoked by a GRANT/REVOKE statement
SYS_PRIVILEGE	System privileges granted /revoked by a GRANT/REVOKE statement
ADMIN_OPTION	Signifies the role or system privilege was granted with ADMIN option
GRANTEE	Username of the grantee specified in a GRANT/REVOKE statement
AUDIT_OPTION	Auditing option set with the AUDIT statement
SES_ACTIONS	Session summary (a string of 11 characters, one for each action type, in the order Alter, Audit, Comment, Delete, Grant, Index, Insert, Lock, Rename, Select, and Update; coded - for none, S for success, F for failure, and B for both)

	LOGOFF_TIME	Timestamp for user logoff
	LOGOFF_LREAD	Logical reads for the session
	LOGOFF_PREAD	Physical reads for the session
	LOGOFF_LWRITE	Logical writes for the session
	LOGOFF_DLOCK	Deadlocks detected during the session
	COMMENT_TEXT	Text comment on the audit trail entry (inserted by an application program)
	SESSIONID	Numeric ID for each ORACLE session
	ENTRYID	Numeric ID for each audit trail entry in the session
	STATEMENTID	Numeric ID for each statement run (a statement may cause many actions)
	RETURNCODE	ORACLE message code generated by the action (zero if the action succeeded)
	PRIV_USED	System privilege used to execute the action
	OBJECT_LABEL	Optional Trusted ORACLE label associated with the object being audited
	SESSION_LABEL	Trusted ORACLE label associated with the user session
DBA_BLOCKERS	All sessions that have someone waiting on a lock they hold that are not themselves waiting on a lock	
	SESSION_ID	Session holding a lock
DBA_CATALOG	All database tables, views, synonyms, and sequences	
	OWNER	Owner of the object
	TABLE_NAME	Name of the object
	TABLE_TYPE	Type of the object

DBA_CLUSTERS	Description of all clusters in the database	
	OWNER	Owner of the cluster
	CLUSTER_NAME	Name of the cluster
	TABLESPACE_NAME	Name of the tablespace containing the cluster
	PCT_FREE	Minimum percentage of free space in a block
	PCT_USED	Minimum percentage of used space in a block
	KEY_SIZE	Estimated size of cluster key plus associated rows
	INI_TRANS	Initial number of transactions
	MAX_TRANS	Maximum number of transactions
	INITIAL_EXTENT	Size of the initial extent in bytes
	NEXT_EXTENT	Size of secondary extents in bytes
	MIN_EXTENTS	Minimum number of extents allowed in the segment
	MAX_EXTENTS	Maximum number of extents allowed in the segment
	PCT_INCREASE	Percentage increase in extent size
	AVG_BLOCKS_PER_KEY	Number of blocks in the table divided by number of hash keys
	CLUSTER_TYPE	Type of cluster: B-tree index or hash
	FUNCTION	Hash function used for a hash cluster
	HASHKEYS	Number of hash keys for a hash cluster
DBA_CLU_COLUMNS	Mapping of table columns to cluster columns	
	OWNER	Owner of the cluster
	CLUSTER_NAME	Cluster name
	CLU_COLUMN_NAME	Key column in the cluster
	TABLE_NAME	Clustered table name
	TAB_COLUMN_NAME	Key column in the table
DBA_COL_COMMENTS	Comments on columns of all tables and views	
	OWNER	Name of the owner of the object
	TABLE_NAME	Name of the object
	COLUMN_NAME	Name of the column
	COMMENTS	Comment on the object

DBA_COL_PRIVS	All grants on columns in the database	
	GRANTEE	Name of the user to whom access was granted
	OWNER	Username of the owner of the object
	TABLE_NAME	Name of the object
	COLUMN_NAME	Name of the column
	GRANTOR	Name of the user who performed the grant
	PRIVILEGE	Privileges on the column
DBA_CONSTRAINTS	GRANTABLE	YES if the privilege was granted with ADMIN OPTION, otherwise NO
	Constraint definitions on all tables in the database	
	OWNER	Owner of the constraint definition
	CONSTRAINT_NAME	Name associated with the constraint
	CONSTRAINT_TYPE	Type of constraint: C (check constraint on a table), P (primary key), U (unique key), R (referential integrity), V (with check option, on a view)
	TABLE_NAME	Name of table associated with the constraint
	SEARCH_CONDITION	Text of search condition for table check
	R_OWNER	Owner of table used in referential constraint
	R_CONSTRAINT_NAME	Name of unique constraint definition for referenced table
	DELETE_RULE	The delete rule for a referential constraint CASCADE / NO ACTION
	STATUS	Status of constraint ENABLED or DISABLED
DBA_CONS_COLUMNS	Information about all columns in constraint definitions	
	OWNER	Owner of the constraint definition
	CONSTRAINT_NAME	Name associated with the constraint definition
	TABLE_NAME	Name associated with table with constraint definition
	COLUMN_NAME	Name associated with column specified in the constraint definition
	POSITION	Original position of column in definition

DBA_DATA_FILES**Information about database files**

FILE_NAME	Name of the data file
FILE_ID	ID of the data file
TABLESPACE_NAME	Name of the tablespace to which the file belongs
BYTES	Size of the file in bytes
BLOCKS	Size of the file in ORACLE blocks
STATUS	File status: INVALID, AVAILABLE

DBA_DB_LINKS**All database links in the database**

OWNER	Owner of the database link
DB_LINK	Name of the database link
USERNAME	Name user should use when logging on
PASSWORD	Password for logon
HOST	SQL*Net string for connect
CREATED	Creation time of the database link

DBA_DDL_LOCKS**All DDL locks held in the database and all outstanding requests for a DML lock**

SESSION_ID	Session Identifier
OWNER	Owner of the lock
NAME	Name of the lock
TYPE	Lock type: Cursor, Table/Procedure, Body, Trigger, Index, Cluster
MODE_HELD	Lock mode: None, Null, Share, Exclusive
MODE_REQUESTED	Lock request type: None, Null, Share, Exclusive

DBA_DEPENDENCIES**Dependencies to and from all objects**

OWNER	Owner of the object
NAME	Name of the object
TYPE	Type of object: PROCEDURE, PACKAGE, FUNCTION, PACKAGE BODY
REFERENCED_OWNER	Owner of the parent object
REFERENCED_NAME	Name of the parent object
REFERENCED_TYPE	Type of the parent object: PROCEDURE, PACKAGE, FUNCTION, PACKAGE BODY
REFERENCED_LINK_NAME	Name of the link to the parent object (if remote)

DBA_DML_LOCKS**All DML locks held in the database and all outstanding requests for a DML lock**

SESSION_ID	Session holding or acquiring the lock
OWNER	Owner of the lock
NAME	Name of the lock
MODE_HELD	Lock mode: see the table below
MODE_REQUESTED	Lock request type: see the table below

Lock Mode

ROW-S (SS)

ROW-X (SX)

SHARE

S/ROW-X (SSX)

EXCLUSIVE

NONE

Meaning

Row share

Row exclusive

Share

Share row exclusive

Exclusive

MODE_HELD: Lock requested, not yet obtained

MODE_REQUESTED: Lock identifier obtained,
lock not held or requested**DBA_ERRORS****Current errors on all stored objects in the database**

OWNER	Owner of the object
NAME	Name of the object
TYPE	Type of object: PROCEDURE, PACKAGE, FUNCTION, PACKAGE BODY
SEQUENCE	Sequence number, for ordering
LINE	Line number at which this error occurs
POSITION	Position in the line at which this error occurs
	Text of the error

DBA_EXP_FILES**Description of export files**

EXP_VERSION	Version number of the last export
EXP_TYPE	Type of export file (full, cumulative, or incremental)
FILE_NAME	Name of the export file
USER_NAME	Name of user who executed export
TIMESTAMP	Timestamp of the last export

DBA_EXP_OBJECTS	Objects that have been incrementally exported	
	OWNER	Owner of the exported object
	OBJECT_NAME	Name of the exported object
	OBJECT_TYPE	Type of the exported object
	CUMULATIVE	Date of the last cumulative export
	INCREMENTAL	Date of the last incremental export
DBA_EXP_VERSION	EXPORT_VERSION	Version number of the last export session
	Version number of the last export session	
DBA_EXTENTS	EXP_VERSION	Version number of the last export session
	Extents of all segments in the database	
	OWNER	Owner of the segment associated with the extent
	SEGMENT_NAME	Name of the segment associated with the extent
	SEGMENT_TYPE	Type of the segment
	TABLESPACE_NAME	Name of the tablespace containing the extent
	EXTENT_ID	Extent number in the segment
	FILE_ID	Name of the file containing the extent
	BLOCK_ID	Starting block number of the extent
	BYTES	Size of the extent in bytes
DBA_FREE_SPACE	BLOCKS	Size of the extent in ORACLE blocks
	Free extents in all tablespaces	
	TABLESPACE_NAME	Name of the tablespace containing the extent
	FILE_ID	ID number of the file containing the extent
	BLOCK_ID	Starting block number of the extent
	BYTES	Size of the extent in bytes
	BLOCKS	Size of the extent in ORACLE blocks

DBA_INDEXES**Description for all indexes in the database**

OWNER	Username of the owner of the index
STATUS	State of the index: DIRECT LOAD or VALID
INDEX_NAME	Name of the index
TABLE_OWNER	Owner of the indexed object
TABLE_NAME	Name of the indexed object
TABLE_TYPE	Type of the indexed object
UNIQUENESS	Uniqueness status of the index: UNIQUE or NONUNIQUE
TABLESPACE_NAME	Name of the tablespace containing the index
INI_TRANS	Initial number of transactions
MAX_TRANS	Maximum number of transactions
INITIAL_EXTENT	Size of the initial extent
NEXT_EXTENT	Size of secondary extents
MIN_EXTENTS	Minimum number of extents allowed in the segment
MAX_EXTENTS	Maximum number of extents allowed in the segment
PCT_INCREASE	Percentage increase in extent size
PCT_FREE	Minimum percentage of free space in a block
BLEVEL	Depth of the index from its root block to its leaf blocks. A depth of 1 indicates that the root block and leaf block are the same.
LEAF_BLOCKS	Number of leaf blocks in the index.
DISTINCT_KEYS	Number of distinct indexed values. For indexes that enforce UNIQUE and PRIMARY KEY constraints, this value is the same as the number of rows in the table (USER_TABLES.NUM_ROWS).
AVG_LEAF_BLOCKS_ PER_KEY	Average number of leaf blocks in which each distinct value in the index appears. This statistic is rounded to the nearest integer. For indexes that enforce UNIQUE and PRIMARY KEY constraints, this value is always 1.
AVG_DATA_BLOCKS_ PER_KEY	The average number of data blocks in the table that are pointed to by a distinct value in the index. This statistic is the average number of data blocks that contain rows that contain a given value for the indexed

	column(s). This statistic is rounded to the nearest integer.
CLUSTERING_FACTOR	This statistic represents the amount of order of the rows in the table based on the values of the index. If its value is near the number of blocks, then the table is very well ordered. In such a case, the index entries in a single leaf block tend to point to rows in the same data blocks. If its value is near the number of rows, then the table is very randomly ordered. In such a case, it is unlikely that index entries in the same leaf block point to rows in the same data blocks.

DBA_IND_COLUMNS

Columns of indexes on all tables and clusters

INDEX_OWNER	Index owner
INDEX_NAME	Index name
TABLE_OWNER	Table or cluster owner
TABLE_NAME	Table or cluster name
COLUMN_NAME	Column name
COLUMN_POSITION	Position of column within index
COLUMN_LENGTH	Indexed length of the column

DBA_LOCKS

All locks or latches held in the database, and all outstanding requests for a lock or latch. This view includes DML locks and DDL locks.

SESSION_ID	Session holding or acquiring the lock
TYPE	Lock type: See tables beginning on page B-81
MODE_HELD	Lock mode: see table on page B-29
MODE_REQUESTED	Lock request type: see table on page B-29
LOCK_ID1	Type-specific lock identifier, part 1
LOCK_ID2	Type-specific lock identifier, part 2

DBA_OBJECTS**All clusters, database links, indexes, packages, package bodies, sequences, synonyms, tables, and views defined in the database**

OWNER	Username of the owner of the object
OBJECT_NAME	Name of the object
OBJECT_ID	Object number of the object
OBJECT_TYPE	Type of the object
CREATED	Timestamp for the creation of the object
LAST_DDL_TIME	Timestamp for the last modification of the object resulting from a DDL command (including grants and revokes)
TIMESTAMP	Timestamp for the creation of the object (character data)
STATUS	Status of the object: VALID, INVALID, or N/A

DBA_OBJECT_SIZE**All PL/SQL objects in the database**

NAME	Name of the object
TYPE	Type of the object: PROCEDURE, PACKAGE, or PACKAGE BODY
SOURCE_SIZE	Size of source code in bytes
PARSED_SIZE	Size of parsed code in bytes
CODE_SIZE	Size of compiled code in bytes
ERROR_SIZE	Size of error messages in bytes

DBA_OBJ_AUDIT_OPTS	Auditing options for all tables and views	
OWNER	OWNER	Owner of the object
OBJECT_NAME	OBJECT_NAME	Name of the object
OBJECT_TYPE	OBJECT_TYPE	Type of the object: TABLE or VIEW
ALT	ALT	Auditing ALTER WHENEVER SUCCESSFUL / UNSUCCESSFUL
AUD	AUD	Auditing AUDIT WHENEVER SUCCESSFUL / UNSUCCESSFUL
COM	COM	Auditing COMMENT WHENEVER SUCCESSFUL / UNSUCCESSFUL
DEL	DEL	Auditing DELETE WHENEVER SUCCESSFUL / UNSUCCESSFUL
GRA	GRA	Auditing GRANT WHENEVER SUCCESSFUL / UNSUCCESSFUL
IND	IND	Auditing INDEX WHENEVER SUCCESSFUL / UNSUCCESSFUL
INS	INS	Auditing INSERT WHENEVER SUCCESSFUL / UNSUCCESSFUL
LOC	LOC	Auditing LOCK WHENEVER SUCCESSFUL / UNSUCCESSFUL
REN	REN	Auditing RENAME WHENEVER SUCCESSFUL / UNSUCCESSFUL
SEL	SEL	Auditing SELECT WHENEVER SUCCESSFUL / UNSUCCESSFUL
UPD	UPD	Auditing UPDATE WHENEVER SUCCESSFUL / UNSUCCESSFUL
REF	REF	Auditing REFERENCES WHENEVER SUCCESSFUL / UNSUCCESSFUL
EXE	EXE	Auditing EXECUTE WHENEVER SUCCESSFUL / UNSUCCESSFUL
DBA_PRIV_AUDIT_OPTS	Privilege auditing options (one row for each audited privilege)	
USER_NAME	USER_NAME	Username if auditing by username; NULL for system-wide auditing
PRIVILEGE	PRIVILEGE	Name of the privilege
SUCCESS	SUCCESS	Mode for WHENEVER SUCCESSFUL system auditing: BY SESSION, BY ACCESS
FAILURE	FAILURE	Mode for WHENEVER NOT SUCCESSFUL system auditing: BY SESSION, BY ACCESS

DBA_PROFILES	Resource limits assigned to each profile	
	PROFILE	Name of the profile
	RESOURCE_NAME	Name of the resource in the profile
	LIMIT	Limit placed on this resource for this profile
DBA_ROLES	All roles that exist in the database	
	ROLE	Name of the granted role
	PASSWORD_REQUIRED	Indicates if the role requires a password to be enabled. Values are YES (ORACLE password), NO (no authentication), and EXTERNAL (operating system authentication).
DBA_ROLE_PRIVS	Description of roles granted to users and to roles	
	GRANTEE	Name of the user or role receiving the grant
	GRANTED_ROLE	Name of the granted role
	ADMIN_OPTION	“Y” if granted with ADMIN option
	DEFAULT_ROLE	“Y” if it is the grantee’s default role
DBA_ROLLBACK_SEGS	Description of rollback segments	
	SEGMENT_NAME	Name of the rollback segment
	OWNER	Owner of the rollback segment
	TABLESPACE_NAME	Name of the tablespace containing the rollback segment
	SEGMENT_ID	ID number of the rollback segment
	FILE_ID	ID number of the block containing the segment header
	BLOCK_ID	Starting block number of the extent
	INITIAL_EXTENT	Initial extent size in bytes
	NEXT_EXTENT	Secondary extent size in bytes
	MIN_EXTENTS	Minimum number of extents
	MAX_EXTENTS	Maximum number of extents
	PCT_INCREASE	Percent increase for extent size
	STATUS	Rollback segment status: INVALID, IN USE, AVAILABLE, OFFLINE, NEEDS RECOVERY, PARTLY AVAILABLE
	INSTANCE_NUM	Instance this rollback segment belongs to, or NULL for a single-instance system

DBA_SEGMENTS**Storage allocated for all database segments**

OWNER	Username of the segment owner
SEGMENT_NAME	Name of the segment, if any
SEGMENT_TYPE	Type of segment: TABLE, CLUSTER, INDEX, ROLLBACK, DEFERRED ROLLBACK, TEMPORARY, CACHE
TABLESPACE_NAME	Name of the tablespace containing the segment
HEADER_FILE	ID of the file containing the segment header
HEADER_BLOCK	ID of the block containing the segment header
BYTES	Size of the segment in bytes
BLOCKS	Size of the segment in ORACLE blocks. This value can be used as a guide when re-creating a segment. The size of the new segment will be approximately the same as the size of the old segment, as long as the value of PCTFREE for the new segment is less than or equal to the average amount of free space in the old blocks.
EXTENTS	Number of extents allocated to the segment
INITIAL_EXTENT	Size of the initial extent in ORACLE blocks
NEXT_EXTENT	Size of the next extent to be allocated in ORACLE blocks
MIN_EXTENTS	Minimum number of extents allowed in the segment
MAX_EXTENTS	Maximum number of extents allowed in the segment
PCT_INCREASE	Percent by which to increase the size of the next extent to be allocated
FREELISTS	Number of process freelists allocated to this segment
FREELIST_GROUPS	Number of freelist groups allocated to this segment

DBA_SEQUENCES

Description of all sequences in the database

SEQUENCE_OWNER	Name of the owner of the sequence
SEQUENCE_NAME	Sequence name
MIN_VALUE	Minimum value of the sequence
MAX_VALUE	Maximum value of the sequence
INCREMENT_BY	Value by which sequence is incremented
CYCLE_FLAG	Does sequence wraparound on reaching limit
ORDER_FLAG	Are sequence numbers generated in order
CACHE_SIZE	Number of sequence numbers to cache
LAST_NUMBER	Last sequence number written to disk. If a sequence uses caching, the number written to disk is the last number placed in the sequence cache. This number is likely to be greater than the last sequence number that was used.

DBA_SNAPSHOTS**All snapshots in the database**

OWNER	Owner of the snapshot
NAME	Name of the view used by users and applications for viewing the snapshot
TABLE_NAME	Table the snapshot is stored in (it has an extra column for the master rowid)
MASTER_VIEW	View of the master table, owned by the snapshot owner, used for refreshes
MASTER_OWNER	Owner of the master table
MASTER	Master table that this snapshot copies
MASTER_LINK	Database link name to the master site
CAN_USE_LOG	YES if this snapshot can use a snapshot log; NO if this snapshot is too complex to use a log
LAST_REFRESH	Date and time at the master site of the last refresh
ERROR	Error returned last time an automatic refresh was attempted
TYPE	Type of refresh for all automatic refreshes: COMPLETE, FAST, FORCE
NEXT	Date function used to compute next refresh dates
START_WITH	Date function used to compute next refresh dates
QUERY	Query used to create the snapshot

DBA_SNAPSHOT_LOGS**All snapshot logs in the database**

LOG_OWNER	Owner of the snapshot log
MASTER	Name of the master table which the log records changes of
LOG_TABLE	Log table that holds the rowids and timestamps of rows that changed in the master table
LOG_TRIGGER	After-row trigger on the master that inserts rows into the log
CURRENT_SNAPSHOTS	Date the snapshot of the master was last refreshed

DBA_SOURCE	Text source of all stored objects in the database	
	OWNER	Owner of the object
	NAME	Name of the object
	TYPE	Type of object: PROCEDURE, PACKAGE, FUNCTION, PACKAGE BODY
	LINE	Line number of this line of source
	TEXT	Text source of the stored object
DBA_STMT_AUDIT_OPTS	Describes current system auditing options across the system and by user	
	USER_NAME	Username if auditing by username; NULL for system-wide auditing
	AUDIT_OPTION	Name of the system auditing option
	SUCCESS	Mode for WHENEVER SUCCESSFUL system auditing: BY SESSION, BY ACCESS
	FAILURE	Mode for WHENEVER NOT SUCCESSFUL system auditing: BY SESSION, BY ACCESS
DBA_SYNONYMS	All synonyms in the database	
	OWNER	Username of the owner of the synonym
	SYNONYM_NAME	Name of the synonym
	TABLE_OWNER	Owner of the object referenced by the synonym
	TABLE_NAME	Name of the object referenced by the synonym
	DB_LINK	Name of the database link referenced in a remote synonym
DBA_SYS_PRIVS	Description of system privileges granted to users and to roles	
	GRANTEE	Name of the user or role receiving the grant
	PRIVILEGE	System privilege granted to the user or role
	ADMIN_OPTION	Signifies that the privilege was granted with the ADMIN option

DBA_TABLES**Description of all tables in the database**

OWNER	Owner of the table
TABLE_NAME	Name of the table
TABLESPACE_NAME	Tablespace containing the table
CLUSTER_NAME	Name of the cluster, if any, to which the table belongs
PCT_FREE	Minimum percentage of free space in a block
PCT_USED	Minimum percentage of used space in a block
INI_TRANS	Initial number of transactions
MAX_TRANS	Maximum number of transactions
INITIAL_EXTENT	Size of the initial extent in bytes
NEXT_EXTENT	Size of secondary extents in bytes
MIN_EXTENTS	Minimum number of extents allowed in the segment
MAX_EXTENTS	Maximum number of extents allowed in the segment
PCT_INCREASE	Percentage increase in extent size
BACKED_UP	Has table been backed up since last modification
NUM_ROWS	Number of rows in the table
BLOCKS	Number of data blocks allocated to the table
EMPTY_BLOCKS	Number of data blocks allocated to the table that contain no data
AVG_SPACE	Average amount of free space (in bytes) in a data block allocated to the table
CHAIN_CNT	Number of rows in the table that are chained from one data block to another or that have migrated to a new block, requiring a link to preserve the old rowid
AVG_ROW_LEN	Average row length in the table in bytes

DBA_TABLESPACES**Description of all tablespaces in the database**

TABLESPACE_NAME	Tablespace name
INITIAL_EXTENT	Default initial extent size
NEXT_EXTENT	Default incremental extent size
MIN_EXTENTS	Default minimum number of extents
MAX_EXTENTS	Default maximum number of extents
PCT_INCREASE	Default percent increase for extent size
STATUS	Tablespace status: ONLINE, OFFLINE, or INVALID (tablespace has been dropped)

DBA_TAB_COLUMNS**Columns of all tables, views, and clusters**

OWNER	Owner of the table, view or cluster
TABLE_NAME	Table, view, or cluster name
COLUMN_NAME	Column name
DATA_TYPE	Datatype of the column
DATA_LENGTH	Length of the column in bytes
DATA_PRECISION	Decimal precision for NUMBER datatype; binary precision for FLOAT datatype; NULL for all other datatypes.
DATA_SCALE	Digits to right of decimal point in a number
NULLABLE	Does column allow NULLS? Value is N if there is a NOT NULL constraint on the column or if the column is part of a PRIMARY KEY.
COLUMN_ID	Sequence number of the column as created
DEFAULT_LENGTH	Length of default value for the column
DATA_DEFAULT	Default value for the column
NUM_DISTINCT	Number of distinct values in each column of the table
LOW_VALUE	For tables with more than three rows, the second-lowest and second-highest values in the column. For tables with three rows or fewer, the lowest and highest values. These statistics are expressed in hexadecimal notation for the internal representation of the first 32 bytes of the values.
HIGH_VALUE	

DBA_TAB_COMMENTS	Comments on all tables and views in the database	
	OWNER	Owner of the object
	TABLE_NAME	Name of the object
	TABLE_TYPE	Type of the object
	COMMENTS	Comment on the object
DBA_TAB_PRIVS	All grants on objects in the database	
	GRANTEE	User to whom access was granted
	OWNER	Owner of the object
	TABLE_NAME	Name of the object
	GRANTOR	Name of the user who performed the grant
	PRIVILEGE	Privilege on the object
DBA_TRIGGERS	GRANTABLE	YES if the privilege was granted with ADMIN OPTION, otherwise NO
	Description of all triggers in the database	
	OWNER	Owner of the trigger
	TRIGGER_NAME	Name of the trigger
	TRIGGER_TYPE	Type of the trigger: BEFORE ROW, BEFORE STATEMENT, AFTER ROW, AFTER STATEMENT
	TRIGGERING_EVENT	Statement that fires the trigger: INSERT, UPDATE, DELETE
	TABLE_OWNER	Owner of the table the trigger is defined on
	TABLE_NAME	Table the trigger is defined on
	REFERENCING_NAMES	Names used for referencing OLD and NEW column values from within the trigger
	WHEN_CLAUSE	WHEN clause. Must evaluate to TRUE for TRIGGER_BODY to execute
	STATUS	Whether or not the trigger is enabled: ENABLED or DISABLED
	DESCRIPTION	Trigger description. Useful for re-creating a trigger creation statement.
	TRIGGER_BODY	Statement(s) executed by the trigger when it fires

DBA_TRIGGER_COLS	Shows usage of columns in triggers defined by any user or on any user's table	
	TRIGGER_OWNER	Owner of the trigger
	TRIGGER_NAME	Name of the trigger
	TABLE_OWNER	Owner of the table the trigger is defined on
	TABLE_NAME	Table the trigger is defined on
	COLUMN_NAME	Name of the column used in the trigger
	COLUMN_LIST	Column specified in UPDATE clause: Y/N
DBA_TS_QUOTAS	COLUMN_USAGE	How the column is used in the trigger; all applicable combinations of NEW, OLD, IN, OUT, and IN OUT
	Tablespace quotas for all users	
	TABLESPACE_NAME	Tablespace name
	USERNAME	User with resource rights on the tablespace
	BYTES	Number of bytes charged to the user
	MAX_BYTES	User's quota in bytes or UNLIMITED
	BLOCKS	Number of ORACLE blocks charged to the user
DBA_USERS	MAX_BLOCKS	User's quota in ORACLE blocks or UNLIMITED
	Information about all users of the database	
	USERNAME	Name of the user
	USER_ID	ID number of the user
	PASSWORD	Encrypted password
	DEFAULT_TABLESPACE	Default tablespace for data
	TEMPORARY_TABLESPACE	Default tablespace for temporary tables
DBA_VIEWS	CREATED	User creation date
	PROFILE	Name of resource profile assigned to the user
	Text of all views in the database	
	OWNER	Owner (creator) of the view
	VIEW_NAME	Name of the view
	TEXT_LENGTH	Length of the view text
	TEXT	View text

DBA_WAITERS	All sessions waiting for locks and the session that holds the lock	
	WAITING_SESSION	Session waiting for a lock
	HOLDING_SESSION	Session holding a lock
	TYPE	Lock type: see tables beginning on page B-81
	MODE_HELD	Lock mode: Row-S (SS), Row-X (SX), Share, S/Row-X (SSX), Exclusive
	MODE_REQUESTED	Lock request type: Null, Row-S (SS), Row-X (SX), Share, S/Row-X (SSX), Exclusive
	LOCK_ID1	Type-specific lock identifier, part 1
	LOCK_ID2	Type-specific lock identifier, part 2
DBMS_ALERT_INFO	Table of registered alerts	
	NAME	Name of the alert
	SID	Session ID of a session waiting for this alert
	CHANGED	Boolean flag to indicate that an alert has been signaled. Y: Alert signaled. N: No alert.
	MESSAGE	Optional message passed by signaler
DBMS_LOCK_ALLOCATED	Table of user-allocated locks.	
	NAME	Name of the lock
	LOCKID	Lock identifier number
	EXPIRATION	Planned lock expiration date (updates whenever the allocation procedure is run)
DEPTREE	Object dependency tree. Created by DEPTREE.SQL. For user SYS, this view shows shared cursors (and only shared cursors) that depend on the object. For all other users, it shows objects other than shared cursors. (Other users may access SYS.DEPTREE for information on shared cursors.)	
	NESTED_LEVEL	Nesting level in the dependency tree
	TYPE	Object type
	OWNER	Object owner
	NAME	Object name
	SEQ#	Sequence number in the dependency tree. Used for ordering queries. (See also: the IDEPTREE view)
DICT	Synonym for DICTIONARY	

DICTIONARY	Description of data dictionary tables and views	
	TABLE_NAME	Name of the object
	COMMENTS	Text comment on the object
DICT_COLUMNS	Description of columns in data dictionary tables and views	
	TABLE_NAME	Name of the object that contains the column
	COLUMN_NAME	Name of the column
	COMMENTS	Text comment on the column
ERROR_SIZE	Accessed to create the ALL_OBJECT_SIZE, DBA_OBJECT_SIZE, and USER_OBJECT_SIZE views	
EXCEPTIONS	Information on violations of integrity constraints	
	ROW_ID	Row that caused the violation
	OWNER	Owner of the table
	TABLE_NAME	Name of the table
	CONSTRAINT	Integrity constraint that was violated
GLOBAL_NAME	Contains one row that shows the global name of the current database	
	GLOBAL_NAME	Global name of the database
IDEPTREE	Indented dependency tree. Created by DEPTREE.SQL. Pre-sorted, pretty-print version of DEPTREE.	
	NESTED_LEVEL	Nesting level in the dependency tree
	TYPE	Object type
	OWNER	Object owner
	NAME	Object name
IND	Synonym for USER_INDEXES	
INDEX_HISTOGRAM	Contains information from the VALIDATE INDEX command	
	REPEAT_COUNT	Number of times that one or more index keys is repeated in the table
	KEYS_WITH_REPEAT_COUNT	Number of index keys that are repeated that many times

INDEX_STATS

Stores information from the VALIDATE INDEX command

HEIGHT	Height of the B-tree
BLOCKS	Blocks allocated to the segment
NAME	Name of the index
LF_ROWS	Number of leaf rows (values in the index)
LF_BLKs	Number of leaf blocks in the B-tree
LF_ROWS_LEN	Sum of the lengths of all the leaf rows
LF_BLK_LEN	Usable space in a leaf block
BR_ROWS	Number of branch rows in the B-tree
BR_BLKs	Number of branch blocks in the B-tree
BR_ROWS_LEN	Sum of the lengths of all the branch blocks in the B-tree
BR_BLK_LEN	Usable space in a branch block
DEL_LF_ROWS	Number of deleted leaf rows in the index
DEL_LF_ROWS_LEN	Total length of all deleted rows in the index
DISTINCT_KEYS	Number of distinct keys in the index (may include rows that have been deleted)
MOST_REPEATED_KEY	How many times the most repeated key is repeated (may include rows that have been deleted)
BTREE_SPACE	Total space currently allocated in the B-tree
USED_SPACE	Total space that is currently being used in the B-tree
PCT_USED	Percent of space allocated in the B-tree that is being used
ROWS_PER_KEY	Average number of rows per distinct key (this figure is calculated without consideration of deleted rows)
BLKS_GETS_PER_ACCESS	Expected number of consistent mode block reads per row, assuming that a randomly chosen row is accessed using the index. Used to calculate the number of consistent reads that will occur during an index scan.

LOADER_COL_INT0	SQL* LOADER view used for direct loads
LOADER_CONSTRAINT_INFO	SQL*LOADER view used for direct loads
LOADER_INDCOL_INFO	SQL*LOADER view used for direct loads
LOADER_IND_INFO	SQL*LOADER view used for direct loads
LOADER_PARAM_INFO	SQL*LOADER view used for direct loads
LOADER_TAB_INFO	SQL*LOADER view used for direct loads
LOADER_TRIGGER_INFO	SQL*LOADER view used for direct loads
OBJ	Synonym for USER_OBJECTS
PARSED_PIECES	Accessed to create the ALL_OBJECT_SIZE, DBA_OBJECT_SIZE, and USER_OBJECT_SIZE views.
PARSED_SIZE	Accessed to create the ALL_OBJECT_SIZE, DBA_OBJECT_SIZE, and USER_OBJECT_SIZE views.

PLAN_TABLE	Default table for results of the EXPLAIN PLAN statement. One row for each step in the execution plan. Created by UTLXPLAN.SQL.	
	STATEMENT_ID	Optional statement identifier specified in the EXPLAIN PLAN statement
	TIMESTAMP	Date and time that the EXPLAIN PLAN statement was issued
	REMARKS	Place for comments that can be added to the steps of the execution plan
	OPERATION	Name of the operation performed at this step
	OPTIONS	Options used for the operation performed at this step
	OBJECT_NODE	Name of the database link used to reference the object
	OBJECT_OWNER	Owner of the object
	OBJECT_NAME	Name of the object
	OBJECT_INSTANCE	Numbered position of the object name in the original SQL statement
	OBJECT_TYPE	Descriptive modifier that further describes the type of object
	SEARCH_COLUMNS	Not currently used
	ID	Identification number for this step in the execution plan
	PARENT_ID	ID of the next step that operates on the results of this step
	POSITION	Order of processing for steps with the same parent ID. For cost-based optimization, the value in the first row of the plan is the statement's execution cost. For rule-based optimization, the value is null in the first row.
	OTHER	Additional information on this step.
PUBLIC_DEPENDENCY	Dependencies to and from objects, by object number	
	OBJECT_ID	Object number
	REFERENCED_OBJECT_ID	Referenced object (the parent object)
RESOURCE_COST	Cost for each resource	
	RESOURCE_NAME	Name of the resource
	UNIT_COST	Cost of the resource

ROLE_ROLE_PRIVS	Information about roles granted to other roles	
	ROLE	Name of the role
	GRANTED_ROLE	Role that was granted
	ADMIN_OPTION	Signifies that the role was granted with ADMIN option
ROLE_SYS_PRIVS	Information about system privileges granted to roles	
	ROLE	Name of the role
	PRIVILEGE	System privilege granted to the role
	ADMIN_OPTION	Signifies the grant was with the ADMIN option
ROLE_TAB_PRIVS	Information about table privileges granted to roles	
	ROLE	Name of the role
	OWNER	Owner of the object
	TABLE_NAME	Name of the object
	COLUMN_NAME	Name of the column, if applicable
	PRIVILEGE	Object privilege granted to the role
	GRANTABLE	YES if the role was granted with ADMIN OPTION, otherwise NO
SEQ	Synonym for USER_SEQUENCES	
SESSION_PRIVS	Privileges that are currently available to the user	
	PRIVILEGE	Name of the privilege
SESSION_ROLES	Roles that the user currently has enabled	
	ROLE	Name of the role
SOURCE_SIZE	Accessed to create the ALL_OBJECT_SIZE, DBA_OBJECT_SIZE, and USER_OBJECT_SIZE views.	
STMT_AUDIT_OPTION_MAP	Description table for auditing option type codes	
	OPTION#	Numeric auditing option type code
	NAME	Name of the auditing option
SYN	Synonym for USER_SYNONYMS	
SYSTEM_PRIVILEGE_MAP	Description table for system privilege codes	

TABLE_PRIVILEGES	Grants on objects for which the user is the grantor, grantee, or owner, or PUBLIC is the grantee	
	GRANTEE	Name of the user to whom access is granted
	OWNER	Owner of the object
	TABLE_NAME	Name of the object
	GRANTOR	Name of the user who performed the grant
	SELECT_PRIV	Permission to select from the object
	INSERT_PRIV	Permission to insert into the object
	DELETE_PRIV	Permission to delete from the object
	UPDATE_PRIV	Permission to update the object
	REFERENCES_PRIV	Permission to reference the object
	ALTER_PRIV	Permission to alter the object
	INDEX_PRIV	Permission to create or drop an index on the object
	CREATED	Timestamp for the grant
TABLE_PRIVILEGE_MAP	Description table for access privilege codes	
TABS	Synonym for USER_TABLES	

USER_AUDIT_OBJECT	Audit trail records for statements concerning objects. Created by CATAUDIT.SQL.
OS_USERNAME	Operating system logon username of the user whose actions were audited
USERNAME	Name (not ID number) of the user whose actions were audited
USERHOST	Numeric instance ID for the ORACLE instance from which the user is accessing the database
TERMINAL	Identifier for the user's terminal
TIMESTAMP	Timestamp for the creation of the audit trail entry or logon time for the CONNECT statement
OWNER	Creator of object affected by the action
OBJ_NAME	Name of the object affected by the action
ACTION_NAME	Name of the action type corresponding to the numeric code in ACTION
NEW_OWNER	Owner of the object named in the NEW_NAME column
NEW_NAME	New name of an object renamed by a RENAME statement
SES_ACTIONS	Session summary (a string of 11 characters, one for each action type, in the order Alter, Audit, Comment, Delete, Grant, Index, Insert, Lock, Rename, Select, and Update; coded - for none, S for success, F for failure, and B for both)
COMMENT_TEXT	Text comment on the audit trail entry (inserted by an application program)
SESSIONID	Numeric ID for each ORACLE session
ENTRYID	Numeric ID for each audit trail entry in the session
STATEMENTID	Numeric ID for each statement run (a statement may cause many actions)
RETURNCODE	ORACLE message code generated by the action (zero if the action succeeded)
PRIV_USED	System privilege used to execute the action
OBJECT_LABEL	Optional Trusted ORACLE label associated with the object being audited
SESSION_LABEL	Trusted ORACLE label associated with the user session

USER_AUDIT_SESSION	All audit trail records concerning connections and disconnections for the user. Created by CATAUDIT.SQL.
OS_USERNAME	Operating system logon user name of the user whose actions were audited
USERNAME	Name (not ID number) of the user whose actions were audited
USERHOST	Numeric instance ID for the ORACLE instance from which the user is accessing the database
TERMINAL	Identifier for the user's terminal
TIMESTAMP	Timestamp for the creation of the audit trail entry or logon time for the CONNECT statement
ACTION_NAME	Name of the action type corresponding to the numeric code in ACTION
LOGOFF_TIME	Timestamp for user logoff
LOGOFF_LREAD	Logical reads for the session
LOGOFF_PREAD	Physical reads for the session
LOGOFF_LWRITE	Logical writes for the session
LOGOFF_DLOCK	Deadlocks detected during the session
SESSIONID	Numeric ID for each ORACLE session
RETURNCODE	ORACLE message code generated by the action (zero if the action succeeded)
SESSION_LABEL	Trusted ORACLE label associated with the user session

USER_AUDIT_STATEMENT

Audit trail entries for the following statements issued by the user: GRANT, REVOKE, AUDIT, NOAUDIT, and ALTER SYSTEM. Created by CATAUDIT.SQL.

OS_USERNAME	Operating system logon username of the user whose actions were audited
USERNAME	Name (not ID number) of the user whose actions were audited
USERHOST	Numeric instance ID for the ORACLE instance from which the user is accessing the database
TERMINAL	Identifier for the user's terminal
TIMESTAMP	Timestamp for the creation of the audit trail entry or logon time for the CONNECT statement
OWNER	Creator of object affected by the action
OBJ_NAME	Name of the object affected by the action
ACTION_NAME	Name of the action type corresponding to the numeric code in ACTION
NEW_NAME	New name of an object after a RENAME
OBJ_PRIVILEGE	Object privileges granted/revoked by a GRANT/REVOKE statement
SYS_PRIVILEGE	System privileges granted/revoked by a GRANT/REVOKE statement
ADMIN_OPTION	Signifies the role or system privilege was granted with ADMIN option
GRANTEE	Username of the grantee specified in a GRANT/REVOKE statement
AUDIT_OPTION	Auditing option set with the AUDIT statement
SES_ACTIONS	Session summary (a string of 11 characters, one for each action type, in the order Alter, Audit, Comment, Delete, Grant, Index, Insert, Lock, Rename, Select, and Update; coded - for none, S for success, F for failure, and B for both)
COMMENT_TEXT	Text comment on the audit trail entry (inserted by an application program)
SESSIONID	Numeric ID for each ORACLE session
ENTRYID	Numeric ID for each audit trail entry in the session
STATEMENTID	Numeric ID for each statement run (a statement may cause many actions)

	RETURNCODE	ORACLE message code generated by the action (zero if the action succeeded)
	PRIV_USED	System privilege used to execute the action
	SESSION_LABEL	Trusted ORACLE label associated with the user session
USER_AUDIT_TRAIL	Audit trail entries relevant to the user. Created by CATAUDIT.SQL.	
	OS_USERNAME	Operating system logon username of the user whose actions were audited
	USERNAME	Name (not ID number) of the user whose actions were audited
	USERHOST	Numeric instance ID for the ORACLE instance from which the user is accessing the database
	TERMINAL	Identifier for the user's terminal
	TIMESTAMP	Timestamp for the creation of the audit trail entry or logon time for the CONNECT statement
	OWNER	Creator of object affected by the action
	OBJ_NAME	Name of the object affected by the action
	ACTION	Numeric type code corresponding to the action name
	ACTION_NAME	Name of the action type corresponding to the numeric code in ACTION
	NEW_OWNER	Owner of the object named in the NEW_NAME column
	NEW_NAME	New name of an object renamed by a RENAME statement
	OBJ_PRIVILEGE	Object privileges granted / revoked by a GRANT/REVOKE statement
	SYS_PRIVILEGE	System privileges granted / revoked by a GRANT/REVOKE statement
	ADMIN_OPTION	Signifies the role or system privilege was granted with ADMIN option
	GRANTEE	Username of the grantee specified in a GRANT/REVOKE statement
	AUDIT_OPTION	Auditing option set with the AUDIT statement
	SES_ACTIONS	Session summary (a string of 11 characters, one for each action type, in the order Alter, Audit, Comment, Delete, Grant, Index, Insert, Lock, Rename, Select, and Update; coded - for none, S for success, F for failure, and B for both)

	LOGOFF_TIME	Timestamp for user logoff
	LOGOFF_LREAD	Logical reads for the session
	LOGOFF_PREAD	Physical reads for the session
	LOGOFF_LWRITE	Logical writes for the session
	LOGOFF_DLOCK	Deadlocks detected during the session
	COMMENT_TEXT	Text comment on the audit trail entry (inserted by an application program)
	SESSIONID	Numeric ID for each ORACLE session
	ENTRYID	Numeric ID for each audit trail entry in the session
	STATEMENTID	Numeric ID for each statement run (a statement can cause many actions)
	RETURNCODE	ORACLE message code generated by the action (zero if the action succeeded)
	PRIV_USED	System privilege used to execute the action
	OBJECT_LABEL	Optional Trusted ORACLE label associated with the object being audited
	SESSION_LABEL	Trusted ORACLE label associated with the user session
USER_CATALOG	Tables, views, synonyms, and sequences owned by the user	
	TABLE_NAME	Name of the object
	TABLE_TYPE	Type of the object

USER_CLUSTERS**Description of user's own clusters**

CLUSTER_NAME	Name of the cluster
TABLESPACE_NAME	Name of the tablespace containing the cluster
PCT_FREE	Minimum percentage of free space in a block
PCT_USED	Minimum percentage of used space in a block
KEY_SIZE	Estimated size of cluster key plus associated rows
INI_TRANS	Initial number of transactions
MAX_TRANS	Maximum number of transactions
INITIAL_EXTENT	Size of the initial extent in bytes
NEXT_EXTENT	Size of secondary extents in bytes
MIN_EXTENTS	Minimum number of extents allowed in the segment
MAX_EXTENTS	Maximum number of extents allowed in the segment
PCT_INCREASE	Percentage increase in extent size
AVG_BLOCKS_PER_KEY	Number of blocks in the table divided by number of hash keys
CLUSTER_TYPE	Type of cluster: B-tree index or hash
FUNCTION	Hash function used for a hash cluster
HASHKEYS	Number of hash keys for a hash cluster

USER_CLU_COLUMNS**Mapping of columns in user's tables to cluster columns**

CLUSTER_NAME	Cluster name
CLU_COLUMN_NAME	Key column in the cluster
TABLE_NAME	Clustered table name
TAB_COLUMN_NAME	Key column in the table

USER_COL_COMMENTS**Comments on columns of user's tables and views**

TABLE_NAME	Object name
COLUMN_NAME	Column name
COMMENTS	Comment on the column

USER_COL_PRIVS	Grants on columns for which the user is the owner, grantor, or grantee	
	GRANTEE	Name of the user to whom access was granted
	OWNER	Owner of the object
	TABLE_NAME	Name of the object
	COLUMN_NAME	Name of the column
	GRANTOR	Name of the user who performed the grant
	PRIVILEGE	Privilege on the column
USER_COL_PRIVS_MADE	All grants on columns of objects owned by the user	
	GRANTEE	Name of the user to whom access was granted
	TABLE_NAME	Name of the object
	COLUMN_NAME	Name of the column
	GRANTOR	Name of the user who performed the grant
	PRIVILEGE	Privilege on the column
	GRANTABLE	YES if the privilege was granted with ADMIN OPTION; otherwise NO
USER_COL_PRIVS_RECD	Grants on columns for which the user is the grantee	
	OWNER	Username of the owner of the object
	TABLE_NAME	Name of the object
	COLUMN_NAME	Name of the column
	GRANTOR	Name of the user who performed the grant
	PRIVILEGE	Privilege on the column
	GRANTABLE	YES if the privilege was granted with ADMIN OPTION; otherwise NO

USER_CONSTRAINTS	Constraint definitions on user's tables	
	OWNER	Owner of the constraint definition
	CONSTRAINT_NAME	Name associated with the constraint definition
	CONSTRAINT_TYPE	Type of constraint definition: C (check constraint on a table), P (primary key), U (unique key), R (referential integrity), V (with check option, on a view)
	TABLE_NAME	Name associated with table with constraint definition
	SEARCH_CONDITION	Text of search condition for table check
	R_OWNER	Owner of table used in referential constraint
	R_CONSTRAINT_NAME	Name of unique constraint definition for referenced table
	DELETE_RULE	The delete rule for a referential constraint: CASCADE, NO ACTION
	STATUS	Status of constraint: ENABLED, DISABLED
USER_CONS_COLUMNS	Information about columns in constraint definitions owned by the user	
	OWNER	Owner of the constraint definition
	CONSTRAINT_NAME	Name associated with the constraint definition
	TABLE_NAME	Name associated with table with constraint definition
	COLUMN_NAME	Name associated with column specified in the constraint definition
	POSITION	Original position of column in definition
USER_DB_LINKS	Database links owned by the user	
	DB_LINK	Name of the database link
	USERNAME	Name of user to log on as
	PASSWORD	Password for logon
	HOST	SQL*Net string for connect
	CREATED	Creation time of the database link

USER_DEPENDENCIES	Dependencies to and from a user's objects	
	NAME	Name of the object
	TYPE	Type of object: PROCEDURE, PACKAGE, FUNCTION, PACKAGE BODY
	REFERENCED_OWNER	Owner of the parent object
	REFERENCED_NAME	Name of the parent object
	REFERENCED_TYPE	Type of the parent object: PROCEDURE, PACKAGE, FUNCTION, PACKAGE BODY
	REFERENCED_LINK_NAME	Name of the link to the parent object (if remote)
USER_ERRORS	Current errors on all a user's stored objects	
	NAME	Name of the object
	TYPE	Type of object: PROCEDURE, PACKAGE, FUNCTION, PACKAGE BODY
	SEQUENCE	Sequence number, for ordering
	LINE	Line number at which this error occurs
	POSITION	Position in the line at which this error occurs
	TEXT	Text of the error
USER_EXTENTS	Extents of the segments belonging to a user's objects	
	SEGMENT_NAME	Name of the segment associated with the extent
	SEGMENT_TYPE	Type of the segment
	TABLESPACE_NAME	Name of the tablespace containing the extent
	EXTENT_ID	Extent number in the segment
	BYTES	Size of the extent in bytes
	BLOCKS	Size of the extent in ORACLE blocks
USER_FREE_SPACE	Free extents in tablespaces accessible to the user	
	TABLESPACE_NAME	Name of the tablespace containing the extent
	FILE_ID	ID number of the file containing the extent
	BLOCK_ID	Starting block number of the extent
	BYTES	Size of the extent in bytes
	BLOCKS	Size of the extent in ORACLE blocks

USER_INDEXES

Description of the user's own indexes

STATUS	State of the index: DIRECT LOAD or VALID
INDEX_NAME	Name of the index
TABLE_OWNER	Owner of the indexed object
TABLE_NAME	Name of the indexed object
TABLE_TYPE	Type of the indexed object
UNIQUENESS	Uniqueness status of the index: UNIQUE or NONUNIQUE
TABLESPACE_NAME	Name of the tablespace containing the index
INI_TRANS	Initial number of transactions
MAX_TRANS	Maximum number of transactions
INITIAL_EXTENT	Size of the initial extent in bytes
NEXT_EXTENT	Size of secondary extents in bytes
MIN_EXTENTS	Minimum number of extents allowed in the segment
MAX_EXTENTS	Maximum number of extents allowed in the segment
PCT_INCREASE	Percentage increase in extent size
PCT_FREE	Minimum percentage of free space in a block
BLEVEL	Depth of the index from its root block to its leaf blocks. A depth of 1 indicates that the root block and leaf block are the same.
LEAF_BLOCKS	Number of leaf blocks in the index.
DISTINCT_KEYS	Number of distinct indexed values. For indexes that enforce UNIQUE and PRIMARY KEY constraints, this value is the same as the number of rows in the table (USER_TABLES.NUM_ROWS).
AVG_LEAF_BLOCKS_ PER_KEY	Average number of leaf blocks in which each distinct value in the index appears. This statistic is rounded to the nearest integer. For indexes that enforce UNIQUE and PRIMARY KEY constraints, this value is always 1.
AVG_DATA_BLOCKS_ PER_KEY	Average number of data blocks in the table that are pointed to by a distinct value in the index. This statistic is the average number of data blocks that contain rows that contain a given value for the indexed column(s). This statistic is rounded to the nearest integer.

	CLUSTERING_FACTOR	This statistic represents the amount of order of the rows in the table based on the values of the index. If its value is near the number of blocks, then the table is very well ordered. In such a case, the index entries in a single leaf block tend to point to rows in the same data blocks. If its value is near the number of rows, then the table is very randomly ordered. In such a case, it is unlikely that index entries in the same leaf block point to rows in the same data blocks.
USER_IND_COLUMNS	Columns of the user's indexes or on user's tables	
	INDEX_NAME	Index name
	TABLE_NAME	Table or cluster name
	COLUMN_NAME	Column name
	COLUMN_POSITION	Position of column within index
	COLUMN_LENGTH	Indexed length of the column
USER_OBJECTS	Objects owned by the user	
	OBJECT_NAME	Name of the object
	OBJECT_ID	Object number of the object
	OBJECT_TYPE	Type of the object
	CREATED	Timestamp for the creation of the object
	LAST_DDL_TIME	Timestamp of the last DDL command applied to the object (including grants and revokes)
	TIMESTAMP	Timestamp for the creation of the object (character data)
	STATUS	Status of the object: VALID, INVALID
USER_OBJECT_SIZE	User's PL/SQL objects	
	NAME	Name of the object
	TYPE	Type of the object: PROCEDURE, PACKAGE, or PACKAGE BODY
	SOURCE_SIZE	Size of source code in bytes
	PARSED_SIZE	Size of parsed code in bytes
	CODE_SIZE	Size of compiled code in bytes
	ERROR_SIZE	Size of error messages in bytes

USER_OBJ_AUDIT_OPTS	Auditing options for user's own tables and views. Created by CATAUDIT.SQL.	
	OBJECT_NAME	Name of the object
	OBJECT_TYPE	Type of the object TABLE or VIEW
	ALT	Auditing ALTER WHENEVER SUCCESSFUL / UNSUCCESSFUL
	AUD	Auditing AUDIT WHENEVER SUCCESSFUL / UNSUCCESSFUL
	COM	Auditing COMMENT WHENEVER SUCCESSFUL / UNSUCCESSFUL
	DEL	Auditing DELETE WHENEVER SUCCESSFUL / UNSUCCESSFUL
	GRA	Auditing GRANT WHENEVER SUCCESSFUL / UNSUCCESSFUL
	IND	Auditing INDEX WHENEVER SUCCESSFUL / UNSUCCESSFUL
	INS	Auditing INSERT WHENEVER SUCCESSFUL / UNSUCCESSFUL
	LOC	Auditing LOCK WHENEVER SUCCESSFUL / UNSUCCESSFUL
	REN	Auditing RENAME WHENEVER SUCCESSFUL / UNSUCCESSFUL
	SEL	Auditing SELECT WHENEVER SUCCESSFUL / UNSUCCESSFUL
	UPD	Auditing UPDATE WHENEVER SUCCESSFUL / UNSUCCESSFUL
	REF	Auditing REFERENCES WHENEVER SUCCESSFUL / UNSUCCESSFUL
	EXE	Auditing EXECUTE WHENEVER SUCCESSFUL / UNSUCCESSFUL
USER_RESOURCE_LIMITS	Displays the resource limits for the current user	
	RESOURCE_NAME	Name of the resource
	LIMIT	Limit placed on this resource

USER_ROLE_PRIVS**Roles granted to the user**

USERNAME	Name of the user, or PUBLIC
GRANTED_ROLE	Name of the role granted to user
ADMIN_OPTION	Granted with ADMIN option: Y/N
DEFAULT_ROLE	Role is designated as the user's default role: Y/N
OS_GRANTED	Granted by the operating system: Y/N (occurs if configuration parameter OS_ROLES = TRUE)

USER_SEGMENTS**Storage allocation for database segments belonging to a user's objects**

SEGMENT_NAME	Name of the segment, if any
SEGMENT_TYPE	Type of segment: TABLE, CLUSTER, INDEX, ROLLBACK, DEFERRED ROLLBACK, TEMPORARY, CACHE
TABLESPACE_NAME	Name of the tablespace containing the segment
BYTES	Size of the segment in bytes
BLOCKS	Size of the segment in ORACLE blocks
EXTENTS	Number of extents allocated to the segment
INITIAL_EXTENT	Size of the initial extent in ORACLE blocks
NEXT_EXTENT	Size of the next extent to be allocated in ORACLE blocks
MIN_EXTENTS	Minimum number of extents allowed in the segment
MAX_EXTENTS	Maximum number of extents allowed in the segment
PCT_INCREASE	Percent by which to increase the size of the next extent to be allocated
FREELISTS	Number of process freelists allocated to this segment
FREELIST_GROUPS	Number of freelist groups allocated to this segment

USER_SEQUENCES

Description of the user's own sequences

SEQUENCE_NAME	SEQUENCE name
MIN_VALUE	Minimum value of the sequence
MAX_VALUE	Maximum value of the sequence
INCREMENT_BY	Value by which sequence is incremented
CYCLE_FLAG	Does sequence wraparound on reaching limit
ORDER_FLAG	Are sequence numbers generated in order
CACHE_SIZE	Number of sequence numbers to cache
LAST_NUMBER	Last sequence number written to disk. If a sequence uses caching, the number written to disk is the last number placed in the sequence cache. This number is likely to be greater than the last sequence number that was actually used. This value is <i>not</i> continuously updated during database operation. It is intended for use after a warmstart or import.

USER_SNAPSHOTS	Snapshots the user can view	
	OWNER	Owner of the snapshot
	NAME	Name of the view used by users and applications for viewing the snapshot
	TABLE_NAME	Table the snapshot is stored in. This table has an extra column for the master rowid
	MASTER_VIEW	View of the master table, owned by the snapshot owner, used for refreshes
	MASTER_OWNER	Owner of the master table
	MASTER	Name of the master table that this snapshot is a copy of
	MASTER_LINK	Database link name to the master site
	CAN_USE_LOG	YES if this snapshot can use a snapshot log, NO if this snapshot is too complex to use a log
	LAST_REFRESH	Date and time at the master site of the last refresh
	ERROR	Error returned last time an automatic refresh was attempted
	TYPE	Type of refresh for all automatic refreshes: COMPLETE, FAST, FORCE
	NEXT	Date function used to compute next refresh dates
	START_WITH	Date function used to compute next refresh dates
	QUERY	Original query that this snapshot is an instantiation of
USER_SNAPSHOT_LOGS	All snapshot logs owned by the user	
	LOG_OWNER	Owner of the snapshot log
	MASTER	Name of the master table which the log records changes of
	LOG_TABLE	Log table that holds the rowids and timestamps of rows which changed in the master table
	LOG_TRIGGER	An after-row trigger on the master which inserts rows into the log
	CURRENT_SNAPSHOTS	Date the snapshot of the master was last refreshed

USER_SOURCE	Text source of all stored objects belonging to the user	
	NAME	Name of the object
	TYPE	Type of object PROCEDURE, PACKAGE, FUNCTION, PACKAGE BODY
	LINE	Line number of this line of source
	TEXT	Text source of the stored object
USER_SYNONYMS	The user's private synonyms	
	SYNONYM_NAME	Name of the synonym
	TABLE_OWNER	Owner of the object referenced by the synonym
	TABLE_NAME	Name of the object referenced by the synonym
	DB_LINK	Database link referenced in a remote synonym
USER_SYS_PRIVS	System privileges granted to the user	
	USERNAME	Name of the user, or PUBLIC
	PRIVILEGE	System privilege granted to the user
	ADMIN_OPTION	Signifies the privilege was granted with ADMIN option

USER_TABLES**Description of the user's own tables**

TABLE_NAME	Name of the table
TABLESPACE_NAME	Name of the tablespace containing the table
CLUSTER_NAME	Name of the cluster, if any, to which the table belongs
PCT_FREE	Minimum percentage of free space in a block
PCT_USED	Minimum percentage of used space in a block
INI_TRANS	Initial number of transactions
MAX_TRANS	Maximum number of transactions
INITIAL_EXTENT	Size of the initial extent in bytes
NEXT_EXTENT	Size of secondary extents in bytes
MIN_EXTENTS	Minimum number of extents allowed in the segment
MAX_EXTENTS	Maximum number of extents allowed in the segment
PCT_INCREASE	Percentage increase in extent size
BACKED_UP	Has table been backed up since last modification
NUM_ROWS	Number of rows in the table
BLOCKS	Number of data blocks allocated to the table
EMPTY_BLOCKS	Number of data blocks allocated to the table that contain no data
AVG_SPACE	Average amount of free space (in bytes) in a data block allocated to the table
CHAIN_CNT	Number of rows in the table that are chained from one data block to another or that have migrated to a new block, requiring a link to preserve the old rowid
AVG_ROW_LEN	Average length of a row in the table in bytes

USER_TABLESPACES	Description of accessible tablespaces	
	TABLESPACE_NAME	Tablespace name
	INITIAL_EXTENT	Default initial extent size
	NEXT_EXTENT	Default incremental extent size
	MIN_EXTENTS	Default minimum number of extents
	MAX_EXTENTS	Default maximum number of extents
	PCT_INCREASE	Default percent increase for extent size
	STATUS	Tablespace status: ONLINE, OFFLINE, or INVALID (tablespace has been dropped)
USER_TAB_COLUMNS	Columns of user's tables, views, and clusters	
	TABLE_NAME	Table, view, or cluster name
	COLUMN_NAME	Column name
	DATA_TYPE	Datatype of the column
	DATA_LENGTH	Maximum length of the column in bytes
	DATA_PRECISION	Decimal precision for NUMBER datatype; binary precision for FLOAT datatype; NULL for all other datatypes.
	DATA_SCALE	Digits to right of decimal point in a number
	NULLABLE	Does column allow NULLS? Value is n if there is a NOT NULL constraint on the column or if the column is part of a PRIMARY KEY.
	COLUMN_ID	Sequence number of the column as created
	DEFAULT_LENGTH	Length of default value for the column
	DATA_DEFAULT	Default value for the column
	NUM_DISTINCT	Number of distinct values in each column of the table
	LOW_VALUE	For tables with more than three rows, the second-lowest and second-highest values in the column. For tables with three rows or fewer, the lowest and highest values. These statistics are expressed in hexadecimal notation for the internal representation of the first 32 bytes of the values.
	HIGH_VALUE	
USER_TAB_COMMENTS	Comments on the tables and views owned by the user	
	TABLE_NAME	Name of the object
	TABLE_TYPE	Type of the object TABLE or VIEW
	COMMENTS	Comment on the object

USER_TAB_PRIVS	Grants on objects for which the user is the owner, grantor, or grantee	
	GRANTEE	Name of the user to whom access was granted
	OWNER	Owner of the object
	TABLE_NAME	Name of the object
	GRANTOR	Name of the user who performed the grant
	PRIVILEGE	Privilege on the object
	GRANTABLE	YES if the privileges was granted with ADMIN OPTION; otherwise NO
USER_TAB_PRIVS_MADE	All grants on objects owned by the user	
	GRANTEE	Name of the user to whom access was granted
	TABLE_NAME	Name of the object
	GRANTOR	Name of the user who performed the grant
	PRIVILEGE	Privilege on the object
	GRANTABLE	YES if the privilege was granted with ADMIN OPTION; otherwise NO
USER_TAB_PRIVS_RECD	Grants on objects for which the user is the grantee	
	OWNER	Owner of the object
	TABLE_NAME	Name of the object
	GRANTOR	Name of the user who performed the grant
	PRIVILEGE	Privilege on the object
	GRANTABLE	YES if the privilege was granted with ADMIN OPTION; otherwise NO

USER_TRIGGERS		Description of the user's triggers
	TRIGGER_NAME	Name of the trigger
	TRIGGER_TYPE	Type of the trigger: BEFORE STATEMENT, AFTER STATEMENT, BEFORE ROW, AFTER ROW
	TRIGGERING_EVENT	Statement that fires the trigger: INSERT, UPDATE, DELETE
	TABLE_OWNER	Owner of the table the trigger is defined on
	TABLE_NAME	Table the trigger is defined on
	REFERENCING_NAMES	Names used for referencing OLD and NEW column values from within the trigger
	WHEN_CLAUSE	WHEN clause. Must evaluate to TRUE for TRIGGER_BODY to execute
	STATUS	Whether or not the trigger is enabled: ENABLED or DISABLED
	DESCRIPTION	Trigger description. Useful for re-creating a trigger creation statement.
	TRIGGER_BODY	Statement(s) executed by the trigger when it fires
USER_TRIGGER_COLS		Shows usage of columns in triggers owned by user or on one of user's tables
	TRIGGER_OWNER	Owner of the trigger
	TRIGGER_NAME	Name of the trigger
	TABLE_OWNER	Owner of the table the trigger is defined on
	TABLE_NAME	Table the trigger is defined on
	COLUMN_NAME	Name of the column used in the trigger
	COLUMN_LIST	Column specified in UPDATE clause: Y/N
	COLUMN_USAGE	How the column is used in the trigger. All applicable combinations of NEW, OLD, IN, OUT, and IN OUT.
USER_TS_QUOTAS		Tablespace quotas for the user
	TABLESPACE_NAME	Tablespace name
	BYTES	Number of bytes charged to the user
	MAX_BYTES	User's quota in bytes or UNLIMITED
	BLOCKS	Number of ORACLE blocks charged to the user
	MAX_BLOCKS	User's quota in ORACLE blocks or UNLIMITED

USER_USERS

Information about the current user

USERNAME	Name of the user
USER_ID	ID number of the user
DEFAULT_TABLESPACE	Default tablespace for data
TEMPORARY_TABLESPACE	Default tablespace for temporary tables
CREATED	User creation date

USER_VIEWS

Text of views owned by the user

VIEW_NAME	Name of the view
TEXT_LENGTH	Length of the view text
TEXT	View text

The Dynamic Performance Tables

The ORACLE Server contains a set of underlying “tables” that are maintained by the Server and accessible to the DBA user SYS. These tables are called *dynamic performance tables* because they are continuously updated while a database is open and in use, and their contents relate primarily to performance.

Although these tables appear to be regular database tables, they are not. Like ROWIDs and ROWNUMs, these tables may be selected from, but never updated or altered.

The file CATALOG.SQL contains definitions of the views and public synonyms for the dynamic performance tables. You must run CATALOG.SQL to create these views and synonyms.

Views created on the dynamic performance tables are identified by the prefix V_\$, and public synonyms for these views have the prefix V\$. DBAs or users should only access the V\$ objects, not the V_\$ objects.

In general, these views are not intended for direct access by DBAs or users. Rather, the dynamic performance tables are used by SQL*DBA, which is the primary interface for accessing information about system performance.



Once the instance is started, the V\$ views are accessible. The database does not have to be mounted or open. One important consequence of this fact is that the V\$LOG view can be used to identify logfiles needed for recovery.

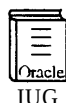


Note: Information about the dynamic performance tables is presented for completeness only; this information does not imply a commitment to support these tables in the future.

Access to the Dynamic Performance Tables

After installation, only username SYS has access to the dynamic performance tables. However, access to these tables is required for any user needing to view the MONITOR displays available in SQL*DBA.

Granting Access to All Tables



The UTLMONTR.SQL script can be run to grant access to PUBLIC on all of the dynamic performance tables. See your installation or user's guide for details on running UTLMONTR.SQL

Granting Access on Selected Tables

If any user other than SYS wants to use SQL*DBA'S MONITOR functions, that user needs access to one or more of the dynamic performance tables. For a list of the views used by each monitor, see the *ORACLE7 Server Utilities User's Guide*.

Descriptions of Individual Tables

V\$ACCESS

Shows objects in the database that are currently locked and the sessions that are accessing them.

SID	VARCHAR2	Session number that is accessing an object
OWNER	VARCHAR2	Owner of the object
OBJECT	VARCHAR2	Name of the object
OB_TYP	NUMBER	Type identifier for the object

V\$ARCHIVE

Information on archive logs for each thread in the database system. (Each row provides information for one thread.)

GROUP#	NUMBER	Logfile group number
THREAD#	NUMBER	Logfile thread number
SEQUENCE#	NUMBER	Logfile sequence number
CURRENT	VARCHAR2	Archive log currently in use
FIRST_CHANGE#	NUMBER	First SCN stored in the current log

V\$BACKUP

Backup status of all online datafiles.

FILE#	NUMBER	File identifier
STATUS	VARCHAR2	File status: NOT ACTIVE, ACTIVE (backup in progress), OFFLINE NORMAL, or description of an error
CHANGE#	NUMBER	System change number when backup started
TIME	NUMBER	Time the backup started

V\$BGPROCESS**Describes the background processes.**

PADDR	RAW(4)	Address of the process state object
NAME	VARCHAR2	Name of this background process
DESCRIPTION	VARCHAR2	Description of the background process
ERROR	NUMBER	Error encountered

V\$CIRCUIT**Information about virtual circuits** (user connections to the database through dispatchers and servers).

CIRCUIT	RAW(4)	Circuit address
DISPATCHER	RAW(4)	Current dispatcher process address
SERVER	RAW(4)	Current server process address
WAITER	RAW(4)	Address of server process that is waiting for the (currently busy) circuit to become available
SADDR	RAW(4)	Address of session bound to the circuit
STATUS	VARCHAR2	Status of the circuit: BREAK (currently interrupted), EOF (about to be removed), OUTBOUND (an outward link to a remote database), NORMAL (normal circuit into the local database)
QUEUE	VARCHAR2	Queue the circuit is currently on: COMMON (on the common queue, waiting to be picked up by a server process), DISPATCHER (waiting for the dispatcher), SERVER (currently being serviced), OUTBOUND (waiting to establish an outbound connection, NONE (idle circuit)
MESSAGE0	NUMBER	Size in bytes of the messages in the first message buffer
MESSAGE1	NUMBER	Size in bytes of the messages in the second message buffer
MESSAGES	NUMBER	Total number of messages that have gone through this circuit
BYTES	NUMBER	Total number of bytes that have gone through this circuit
BREAKS	NUMBER	Total number of breaks (interruptions) for this circuit

V\$DATABASE**Database information from the control file.**

NAME	VARCHAR2	Name of the database
CREATED	VARCHAR2	Creation date
LOG_MODE	VARCHAR2	Archive log mode: NOARCHIVELOG or ARCHIVELOG
CHECKPOINT_CHANGE#	NUMBER	Last SCN checkpointed
ARCHIVE_CHANGE#	NUMBER	Last SCN archived

V\$DATAFILE**Datafile information from the control file.**

FILE#	NUMBER	File identification number
STATUS	VARCHAR2	Type of file (system or user) and its status. Values: OFFLINE, SYSOFF, ONLINE, SYSTEM, RECOVER
CHECKPOINT_CHANGE#	NUMBER	SCN at last checkpoint
BYTES	NUMBER	Size in bytes
NAME	VARCHAR2	Name of the file

V\$DBFILE

All data files making up the database. This view is retained for historical compatibility. Use of V\$DATAFILE is recommended instead.

FILE#	NUMBER	File identifier
NAME	VARCHAR2	Name of file

V\$DB_OBJECT_CACHE

Shows database objects that are cached in the library cache. Objects include tables, indexes, clusters, synonym definitions, PL/SQL procedures and packages, and triggers.

OWNER	VARCHAR2	Owner of the object
NAME	VARCHAR2	Name of the object
DB_LINK	VARCHAR2	Database link name, if any
NAMESPACE	VARCHAR2	Library cache namespace of the object: TABLE/PROCEDURE, BODY, TRIGGER, INDEX, CLUSTER, OBJECT
TYPE	VARCHAR2	Type of the object: INDEX, TABLE, CLUSTER, VIEW, SET, SYNONYM, SEQUENCE, PROCEDURE, FUNCTION, PACKAGE, PACKAGE BODY, TRIGGER, CLASS, OBJECT, USER, DBLINK
SHARABLE_MEM	NUMBER	Amount of sharable memory in the shared pool consumed by the object
LOADS	NUMBER	Number of times the object has been loaded. This count also increases when an object has been invalidated
EXECUTIONS	NUMBER	Total number of times this object has been executed
LOCKS	NUMBER	Number of users currently locking this object
PINS	NUMBER	Number of users currently pinning this object

V\$DISPATCHER**Information on the dispatcher processes.**

NAME	VARCHAR2	Name of the dispatcher process
NETWORK	VARCHAR2	Network protocol supported by this dispatcher. For example, TCP or DECNET.
PADDR	RAW(4)	Process address
STATUS	VARCHAR2	Dispatcher status: WAIT (idle), SEND (sending a message), RECEIVE (receiving a message), CONNECT (establishing a connection), DISCONNECT (handling a disconnect request), BREAK (handling a break), OUTBOUND (establishing an outbound connection)
ACCEPT	VARCHAR2	Whether this dispatcher is accepting new connections: YES, NO
MESSAGES	NUMBER	Number of messages processed by this dispatcher
BYTES	NUMBER	Size in bytes of messages processed by this dispatcher
BREAKS	NUMBER	Number of breaks occurring in this connection
OWNED	NUMBER	Number of circuits owned by this dispatcher
CREATED	NUMBER	Number of circuits created by this dispatcher
IDLE	NUMBER	Total idle time for this dispatcher, in hundredths of a second
BUSY	NUMBER	Total busy time for this dispatcher, in hundredths of a second

V\$ENABLEDPRIVS**Which privileges are enabled.**

PRIV_NUMBER	NUMBER	Numeric identifier of enabled privileges

V\$FILESTAT**Information about file read/write statistics.**

FILE#	NUMBER	Number of the file
PHYRDS	NUMBER	Number of physical reads done
PHYWRTS	NUMBER	Number of physical writes done
PHYBLKRD	NUMBER	Number of physical blocks read
PHYBLKWRT	NUMBER	Number of physical blocks written
READTIM	NUMBER	Time spent doing reads if the parameter TIMED_STATISTICS is TRUE; 0 if FALSE
WRITETIM	NUMBER	Time spent doing writes if the parameter TIMED_STATISTICS is TRUE; 0 if FALSE

V\$FIXED_TABLE**Shows all fixed tables, views, and derived tables in the database.**

NAME	VARCHAR2	Name of the object
OBJECT_ID	NUMBER	Identifier of the fixed object
TYPE	VARCHAR2	Object type: TABLE, VIEW

V\$INSTANCE**State of the current instance.**

KEY	VARCHAR2	Name of state variable, from the table below
VALUE	NUMBER	Value of state variable

Instance State Variable	Value
RESTRICTED MODE	0 (False), 1 (True)
SHUTDOWN PENDING	0 (False), 1 (True)
STARTUP TIME - JULIAN	Start time and date in Julian format
STARTUP TIME - SECONDS	Number of seconds since midnight on the startup date

V\$LATCH**Information about each type of latch.** (The rows of this table and the rows of V\$LATCHNAME correspond one-to-one.)

ADDR	RAW(4)	Address of latch object
LATCH#	NUMBER	Latch number
LEVEL#	NUMBER	Latch level
GETS	NUMBER	Number of times obtained wait
MISSES	NUMBER	Number of times obtained with wait but failed first try
SLEEPS	NUMBER	Number of times slept when wanted wait
IMMEDIATE_GETS	NUMBER	Number of times obtained with no wait
IMMEDIATE_MISSES	NUMBER	Number of times failed to get with no wait

V\$LATCHHOLDER**Information about the current latch holders.**

PID	NUMBER	Identifier of process holding the latch
LADDR	RAW(4)	Latch address

V\$LATCHNAME**The decoded latch names for the latches shown in table V\$LATCH.** (The rows of this table and the rows of V\$LATCH correspond one-to-one.)

LATCH#	NUMBER	Latch number
NAME	VARCHAR2	Latch name

V\$LIBRARYCACHE**Statistics on library cache management.**

NAMESPACE	VARCHAR2	Library cache namespace: SQL AREA, TABLE/PROCEDURE, BODY, TRIGGER, INDEX, CLUSTER, OBJECT, PIPE
GETS	NUMBER	Number of times the system requests handles to library objects belonging to this namespace
GETHITS	NUMBER	Number of times the handles are already allocated in the cache. If the handle is not already allocated, it is a miss. The handle is then allocated and inserted into the cache.
GETHITRATIO	NUMBER	Number of GETHITS divided by GETS. Values close to 1 indicate that most of the handles the system has tried to get are cached.
PINS	NUMBER	Number of times the system issues pin requests for objects in the cache in order to access them
PINHITS	NUMBER	Number of times that objects the system is pinning and accessing are already allocated and initialized in the cache. Otherwise, it is a miss, and the system has to allocate it in the cache and initialize it with data queried from the database or generate the data.
PINIHITRATIO	NUMBER	Number of PINHITS divided by number of PINS. Values close to 1 indicate that most of the objects the system has tried to pin and access have been cached
RELOADS	NUMBER	Number of times that library objects have to be reinitialized and reloaded with data because they have been aged out or invalidated
INVALIDATIONS	NUMBER	Number of times that non-persistent library objects (like shared SQL areas) have been invalidated

V\$LICENSE**Information about license limits.**

SESSIONS_MAX	NUMBER	Maximum number of concurrent user sessions allowed for the instance
SESSIONS_WARNING	NUMBER	Warning limit for concurrent user sessions for the instance
SESSIONS_CURRENT	NUMBER	Current number of concurrent user sessions
SESSION_HIGHWATER	NUMBER	Highest number of concurrent user sessions since the instance started
USERS_MAX	NUMBER	Maximum number of named users allowed for the database

V\$LOADCSTAT

SQL*Loader statistics compiled during the execution of a direct load. These statistics apply to the whole load. Any select against this table results in “no rows returned” since you cannot load data and do a query at the same time.

READ	NUMBER	Number of records read
REJECTED	NUMBER	Number of records rejected
TDISCARD	NUMBER	Total number of discards during the load
NDISCARD	NUMBER	Number of discards from the current file
SAVEDATA	NUMBER	Whether or not save data points are used

V\$LOADTSTAT

SQL*Loader statistics compiled during the execution of a direct load. These statistics apply to the current table. Any select against this table results in “no rows returned” since you cannot load data and do a query at the same time.

LOADED	NUMBER	Number of records loaded
REJECTED	NUMBER	Number of records rejected
FAILWHEN	NUMBER	Number of records that failed to meet any WHEN clause
ALLNULL	NUMBER	Number of records that were completely null and were therefore not loaded
LEFT2SKIP	NUMBER	Number of records yet to skip during a continued load

V\$LOCK

Information about locks and resources. Does not include DDL locks.

ADDR	RAW(4)	Address of lock state object
KADDR	RAW(4)	Address of lock
SID	NUMBER	Identifier of process holding the lock
TYPE	VARCHAR2	Resource type (see following table)
ID1	NUMBER	Resource identifier #1
ID2	NUMBER	Resource identifier #2
LMODE	NUMBER	Lock mode held: 1 (null), 2 (row share), 3 (row exclusive), 4 (share), 5 (share row exclusive), 6 (exclusive)
REQUEST	NUMBER	Lock mode requested (same values as LMODE)

The following locks are obtained by user applications. Any process which is blocking others is likely to be holding one of these locks:

User Lock Type	Description
RW	Row wait enqueue lock
TM	DML enqueue lock
TX	Transaction enqueue lock
UL	User supplied lock

The following system locks are held for extremely short periods of time:

System Lock Type	Description
BL	Buffer hash table instance lock
CF	Cross-instance function invocation instance lock
CI	Control file schema global enqueue lock
CS	Control file schema global enqueue lock
DF	Data file instance lock
DM	Mount/startup db primary/secondary instance lock
DR	Distributed recovery process lock
DX	Distributed transaction entry lock
FI	SGA open-file information lock
FS	File set lock
IR	Instance recovery serialization global enqueue lock
IV	Library cache invalidation instance lock
LA..LP	Library cache lock instance lock (A..P = namespace)
LS	Log start/log switch enqueue lock

System Lock Type	Description
MB	Master buffer hash table instance lock
MM	Mount definition global enqueue lock
MR	Media recovery lock
PA..PZ	Library cache pin instance lock (A..Z = namespace)
QA..QZ	Row cache instance lock (A..Z = cache)
RE	USE_ROW_ENQUEUES enforcement lock
RT	Redo thread global enqueue lock
SC	System commit number instance lock
SH	System commit number high water mark enqueue lock
SN	Sequence number instance lock
SQ	Sequence number enqueue lock
ST	Space transaction enqueue lock
SV	Sequence number value lock
TA	Generic enqueue lock
TD	DDL enqueue lock
TE	Extend-segment enqueue lock
TS	Temporary segment enqueue lock (ID2 = 0)
TS	New block allocation enqueue lock (ID2 = 1)
TT	Temporary table enqueue lock
UN	User name lock
WL	Being-written redo log instance lock
WS	Write-atomic-log-switch global enqueue lock

V\$LOG**Log file information from control file.**

GROUP#	NUMBER	Log group number
THREAD#	NUMBER	Log thread number
SEQUENCE#	NUMBER	Log sequence number
BYTES	NUMBER	Size of the log in bytes
MEMBERS	NUMBER	Number of members in the log group
ARCHIVED	VARCHAR2	Archive status: TRUE, FALSE
FIRST_CHANGE#		
	NUMBER	Lowest SCN in the log
FIRST_TIME	VARCHAR2	Time of first SCN in the log

V\$LOGFILE**Information about redo log files.**

GROUP#	NUMBER	Redo log group identifier number
STATUS	VARCHAR2	Status of this log member: INVALID (file is inaccessible), STALE (file's contents are incomplete), DELETED (file is no longer used), or blank (file is in use)

MEMBER	VARCHAR2	Redo log member name

V\$LOGHIST**Log history information from the control file. This view is retained for historical compatibility. Use of V\$LOG_HISTORY is recommended instead.**

THREAD#	NUMBER	Log thread number
SEQUENCE#	NUMBER	Log sequence number
FIRST_CHANGE#		
	NUMBER	Lowest SCN in the log
FIRST_TIME	VARCHAR2	Time of first SCN in the log
SWITCH_CHANGE#		
	NUMBER	SCN at which the log switch occurred; one more than highest SCN in the log

V\$LOG_HISTORY**Archived log names for all logs in the log history.**

THREAD#	NUMBER	Thread number of the archived log
SEQUENCE#	NUMBER	Sequence number of the archived log
TIME	NUMBER	Time of first entry (lowest SCN) in the log
LOW_CHANGE#		
	NUMBER	Lowest SCN in the log
HIGH_CHANGE#		
	NUMBER	Highest SCN in the log
ARCHIVE_NAME		
	VARCHAR2	Name of file when archived, using the naming convention specified by the LOG_ARCHIVE_FORMAT initialization parameter

V\$NLS_PARAMETERS**Current values of NLS parameters.**

NAME	VARCHAR2	Parameter name: NLS_LANGUAGE, NLS_SORT, NLS_TERRITORY, NLS_CHARACTERSET, NLS_CURRENCY, NLS_ISO_CURRENCY, NLS_NUMERIC_CHARACTERS, NLS_DATE_FORMAT, NLS_DATE_LANGUAGE
VALUE	VARCHAR2	NLS parameter value

V\$OPEN_CURSOR**Cursors that each user session currently has opened and parsed.**

SADDR	RAW	Session address
USER_NAME	VARCHAR2	User thjat is logged into the session
ADDRESS	RAW	Used with HASH_VALUE to uniquely identify the SQL statement being executed in the session
HASH_VALUE	NUMBER	Used with ADDRESS to uniquely identify the SQL statement being executed in the session
SQL_TEXT	VARCHAR2	First 60 characters of the SQL statement that is parsed into the open cursor

V\$PARAMETER**Information about current parameter values.**

NUM	NUMBER	Parameter number
NAME	VARCHAR2	Parameter name
TYPE	NUMBER	Parameter type
VALUE	VARCHAR2	Parameter value
ISDEFAULT	VARCHAR2	Default value in use: TRUE, FALSE

V\$PROCESS**Information about currently active processes.**

ADDR	RAW(4)	Address of process state object
PID	NUMBER	ORACLE process identifier
SPID	VARCHAR2	Operating system process identifier
USERNAME	VARCHAR2	Operating system process username. Any Two-Task user coming across the network has “-T” appended to the username.
SERIAL#	NUMBER	Process serial number
TERMINAL	VARCHAR2	Operating system terminal identifier
PROGRAM	VARCHAR2	Program in progress
BACKGROUND	VARCHAR2	1 for a background process; null for a normal process
LATCHWAIT	VARCHAR2	Address of latch waiting for; null if none

V\$QUEUE**Information on the multi-thread message queues.**

PADDR	RAW(4)	Address of the process that owns the queue
TYPE	VARCHAR2	Type of queue: COMMON (processed by servers), OUTBOUND (used by remote servers), DISPATCHER.
QUEUED	NUMBER	Number of items in the queue
WAIT	NUMBER	Total time that all items in this queue have waited. Divide by TOTALQ for average wait per item.
TOTALQ	NUMBER	Total number of items that have ever been in the queue

V\$RECOVERY_LOG

Archived **logs needed to complete media recovery**. This information is derived from the log history (V\$LOG_HJSTORY). The amount of information available is limited by the setting of the MAX_LOG_HISTORY initialization parameter.

THREAD#	NUMBER	Thread number of the archived log
SEQUENCE#	NUMBER	Sequence number of the archived log
TIME	NUMBER	Time of first entry (lowest SCN) in the log
ARCHIVE_NAME	VARCHAR2	Name of file when archived, using the naming convention specified by the LOG_ARCHIVE_FORMAT initialization parameter

V\$RECOVER_FILE**Status of files needing media recovery.**

FILE#	NUMBER	File identifier number
ONLINE	VARCHAR2	Online status: ONLINE, OFFLINE
ERROR	VARCHAR2	Why the file needs to be recovered: NULL if reason unknown, or OFFLINE NORMAL if recovery not needed
CHANGE#	NUMBER	SCN where recovery must start
TIME	VARCHAR2	Time of SCN where recovery must start

V\$REQDIST**Histogram of request times, divided into 12 buckets, or ranges of time.** The time ranges grow exponentially as a function of the bucket number.

BUCKET	NUMBER	Bucket number: 0..11; the maximum time for each bucket is $(4*2^N)/100$ seconds
COUNT	NUMBER	Count of requests whose total time to complete (excluding wait time) falls in this range.

V\$RESOURCE**Information about resources.**

ADDR	RAW(4)	Address of resource object
TYPE	VARCHAR2	Resource type
ID1	NUMBER	Resource identifier #1
ID2	NUMBER	Resource identifier #2

V\$ROLLNAME**Names of all online rollback segments.**

USN	NUMBER	Rollback (“undo”) segment number
NAME	VARCHAR2	Rollback segment name

V\$ROLLSTAT**Statistics for all online rollback segments.**

USN	NUMBER	Rollback segment number
EXTENTS	NUMBER	Number of rollback extents
RSSIZE	NUMBER	Size in bytes of rollback segment
WRITES	NUMBER	Number of bytes written to rollback segment
XACTS	NUMBER	Number of active transactions
GETS	NUMBER	Number of header gets
WAITS	NUMBER	Number of header waits
OPTSIZE	NUMBER	Optimal size of rollback segment
HWMSIZE	NUMBER	High water mark of rollback segment size
SHRINKS	NUMBER	Number of times rollback segment shrank, eliminating one or more additional extents each time
WRAPS	NUMBER	Number of times rollback segment wraps from one extent to another
EXTENDS	NUMBER	Number of times rollback segment was extended to have a new extent
AVESHRINK	NUMBER	Total size of freed extents divided by number of shrinks
AVEACTIVE	NUMBER	Current average size of active extents, where "active" extents have uncommitted transaction data
STATUS	VARCHAR2	ONLINE if the segment is online, or PENDING OFFLINE if the segment is going offline but some active (distributed) transactions are using the rollback segment. When the transactions complete, the segment goes offline.

V\$ROWCACHE

Statistics for data dictionary activity. (Each row contains statistics for one data dictionary cache.)

CACHE#	NUMBER	Row cache ID number
TYPE	VARCHAR2	Parent or subordinate row cache type
SUBORDINATE	NUMBER	Subordinate set number
PARAMETER	VARCHAR2	Name of the INIT.ORA parameter that determines the number of entries in the data dictionary cache
COUNT	NUMBER	Total number of entries in the cache
USAGE	NUMBER	Number of cache entries that contain valid data
FIXED	NUMBER	Number of fixed entries in the cache
GETS	NUMBER	Total number of requests for information on the data object
GETMISSES	NUMBER	Number of data requests resulting in cache misses
SCANS	NUMBER	Number of scan requests
SCANMISSES	NUMBER	Number of times a scan failed to find the data in the cache
SCANCOMPLETES	NUMBER	For a list of subordinate entries, the number of times the list was scanned completely
MODIFICATIONS	NUMBER	Number of inserts, updates, and deletions
FLUSHES	NUMBER	Number of times flushed to disk

V\$SECONDARY



Trusted

A TRUSTED ORACLE view that lists secondary mounted databases. For details, see the *Trusted ORACLE7 Server Administrator's Guide*.

V\$SESSION**Session information for each current session.**

SADDR	RAW(4)	Session address
SID	NUMBER	Session identifier
SERIAL#	NUMBER	Session serial number. Used to uniquely identify a session's objects. Guarantees that session-level commands are applied to the correct session objects in the event that the session ends and another session begins with the same session ID.
AUDSID	NUMBER	Auditing session ID
PADDR	RAW(4)	Address of the process that owns this session
USER#	NUMBER	ORACLE user identifier
USERNAME	VARCHAR2	ORACLE username
COMMAND	NUMBER	Command in progress; see the table on the next page
TADDR	VARCHAR2	Address of transaction state object
LOCKWAIT	VARCHAR2	Address of lock waiting for; null if none
STATUS	VARCHAR2	Status of the session: ACTIVE, INACTIVE, KILLED
SERVER	VARCHAR2	Server type: DEDICATED, SHARED, PSEUDO, NONE
SCHEMA#	NUMBER	Schema user identifier
SCHEMANAME	VARCHAR2	Schema user name
OSUSER	VARCHAR2	Operating system client user name
PROCESS	VARCHAR2	Operating system client process ID
MACHINE	VARCHAR2	Operating system machine name
TERMINAL	VARCHAR2	Operating system terminal name
PROGRAM	VARCHAR2	Operating system program name
TYPE	VARCHAR2	Session type
SQL_ADDRESS	RAW	Used with SQL_HASH_VALUE to identify the SQL statement that is currently being executed
SQL_HASH_VALUE	NUMBER	Used with SQL_ADDRESS to identify the SQL statement that is currently being executed

<u>Command Number</u>	<u>Command</u>
1	CREATE TABLE
2	INSERT
3	SELECT
4	CREATE CLUSTER
5	ALTER CLUSTER
6	UPDATE
7	DELETE
8	DROP
9	CREATE INDEX
10	DROP INDEX
11	ALTER INDEX
12	DROP TABLE
13	---
14	---
15	ALTER TABLE
16	---
17	GRANT
18	REVOKE
19	CREATE SYNONYM
20	DROP SYNONYM
21	CREATE VIEW
22	DROP VIEW
23	---
24	---
25	---
26	LOCK TABLE
27	NO OPERATION
28	RENAME
29	COMMENT
30	AUDIT
31	NOAUDIT
32	CREATE EXTERNAL DATABASE
33	DROP EXTERNAL DATABASE
34	CREATE DATABASE
35	ALTER DATABASE
36	CREATE ROLLBACK SEGMENT
37	ALTER ROLLBACK SEGMENT
38	DROP ROLLBACK SEGMENT
39	CREATE TABLESPACE
40	ALTER TABLESPACE
41	DROP TABLESPACE
42	ALTER SESSION

Continued...

<u>Command Number</u>	<u>Command</u>
43	ALTER USER
44	COMMIT
45	ROLLBACK
46	SAVEPOINT

V\$SESSION_WAIT

Lists the resources or events that active sessions are waiting for.

SID	NUMBER	Session identifier
SEQ#	NUMBER	Sequence number that uniquely identifies this wait. Incremented for each wait.
EVENT	VARCHAR2	Resource or event the session is waiting for
P1TEXT	VARCHAR2	Description of first additional parameter
P1	VARCHAR2	First additional parameter
P2TEXT	VARCHAR2	Description of second parameter
P2	VARCHAR2	Second additional parameter
P3TEXT	VARCHAR2	Description of third parameter
P3	VARCHAR2	Third additional parameter
WAIT_TIME	NUMBER	A non-zero value is the session's last wait time. A zero value means the session is currently waiting.

V\$SESSTAT

For each current session, the current statistics values.

SID	NUMBER	Session identifier
STATISTIC#	NUMBER	Statistic number (identifier)
VALUE	NUMBER	Statistic value

V\$SESS_IO

I/O statistics for each user session.

SID	NUMBER	Session identifier
BLOCK_GETS	NUMBER	Block gets for this session
CONSISTENT_GETS	NUMBER	Consistent gets for this session
PHYSICAL_READS	NUMBER	Physical reads for this session
BLOCK_CHANGES	NUMBER	Block changes for this session
CONSISTENT_CHANGES	NUMBER	Consistent changes for this session

V\$SGA

Summary information on the System Global Area.

NAME	VARCHAR2	SGA component group
VALUE	NUMBER	Memory size in bytes

V\$SGASTAT

Detailed information on the System Global Area.

NAME	VARCHAR2	SGA component name
BYTES	NUMBER	Memory size in bytes

V\$SHARED_SERVER**Information on the shared server processes.**

NAME	VARCHAR2	Name of the server
PADDR	RAW(4)	Server's process address
STATUS	VARCHAR2	Server status: EXEC (executing SQL), WAIT (ENQ) (waiting for a lock), WAIT (SEND (waiting to send data to user), WAIT (COMMON (idle; waiting for a user request), WAIT (RESET) (waiting for a circuit to reset after a break), QUIT (terminating)
MESSAGES	NUMBER	Number of messages processed
BYTES	NUMBER	Total number of bytes in all messages
BREAKS	NUMBER	Number of breaks
CIRCUIT	RAW(4)	Address of circuit currently being serviced
IDLE	NUMBER	Total idle time in hundredths of a second
BUSY	NUMBER	Total busy time in hundredths of a second
REQUESTS	NUMBER	Total number of requests taken from the common queue in this server's lifetime

V\$SQLAREA

Statistics on the shared cursor cache. Each row has statistics on one shared cursor.

SQL_TEXT	VARCHAR2	Text of the SQL statement requiring the cursor, or the PL/SQL anonymous code
SHARABLE_MEM	NUMBER	Amount of memory in bytes that is sharable between users
PERSISTENT_MEM	NUMBER	Amount of per-user memory in bytes that persists for the life of the cursor
RUNTIME_MEM	NUMBER	Amount of per-user memory, in bytes, that is needed only during execution
SORTS	NUMBER	Number of sorts performed by the SQL statement.
VERSION_COUNT	NUMBER	Number of different versions of this cursor. The same SQL text might be used by different users, each on their own version of a table (for example, "SELECT * from EMP" by SCOTT and JONES, when they each have their own version of EMP). In that case, multiple versions of the cursor would exist.
LOADED_VERSIONS	NUMBER	Versions of the cursor that are currently fully loaded, with no parts aged out
OPEN_VERSIONS	NUMBER	Number of versions that some user has an open cursor on.
USERS_OPENING	NUMBER	Number of users that currently have this SQL statement parsed in an open cursor
EXECUTIONS	NUMBER	Total number of times this SQL statement has been executed
USERS_EXECUTING	NUMBER	Number of users that are currently executing this cursor
LOADS	NUMBER	Number of times the cursor has been loaded. after the body of the cursor has been aged out of the cache while the text of the SQL statement remained in it, or after the cursor is invalidated
FIRST_LOAD_TIME	VARCHAR2	Time at which the cursor was first loaded into the SGA

INVALIDATIONS	NUMBER	Number of times the contents of the cursor have been invalidated. For example: because tables referenced in the cursor are dropped, validated, or indexed
PARSE_CALLS	NUMBER	Number of times users executed a parse call for this cursor
DISK_READS	NUMBER	Number of disk blocks read by this cursor and all cursors caused to be executed by this cursor
BUFFER_GETS	NUMBER	Number of buffers gotten (in any mode) by this cursor and all cursors caused to be executed by this cursor
ADDRESS	RAW	Used together with HASH_VALUE to select the full text of the SQL statement from V\$SQLTEXT
HASH_VALUE	NUMBER	Used together with ADDRESS to select the full text of the SQL statement from V\$SQLTEXT

V\$SQLTEXT

The text of SQL statements belonging to shared SQL cursors in the SGA.

ADDRESS	RAW	Used with HASH_VALUE to uniquely identify a cached cursor
HASH_VALUE	NUMBER	Used with ADDRESS to uniquely identify a cached cursor
PIECE	NUMBER	Number used to order the pieces of SQL text
SQL_TEXT	VARCHAR2	A column containing one piece of the SQL text

V\$STATNAME

Decoded statistic names for the statistics shown in the V\$SESSTAT table.

STATISTIC#	NUMBER	Statistic number
NAME	VARCHAR2	Statistic name
CLASS	NUMBER	Statistic class : 1 (User), 2 (Redo), 4 (Enqueue), 8 (Cache)

V\$SYSLABEL



Trusted

A TRUSTED ORACLE view that lists system labels. For details, see the *Trusted ORACLE7 Server Administrator's Guide*.

V\$SYSSTAT**The current system-wide value for each statistic in table V\$SESSTAT.**

STATISTIC#	NUMBER	Statistic number
NAME	VARCHAR2	Statistic name
CLASS	NUMBER	Statistic class: 1 (User), 2 (Redo), 4 (Enqueue), 8 (Cache)
VALUE	NUMBER	Statistic value

V\$THREAD**Thread information from the control file.**

THREAD#	NUMBER	Thread number
STATUS	VARCHAR2	Thread status: OPEN, CLOSED
ENABLED	VARCHAR2	Enabled status: DISABLED, (enabled) PRIVATE, or (enabled) PUBLIC
GROUPS	NUMBER	Number of log groups assigned to this thread
INSTANCE	VARCHAR2	Instance name, if available
OPEN_TIME	VARCHAR2	Last time the thread was opened
CURRENT_GROUP#	NUMBER	Current log group
SEQUENCE#	NUMBER	Sequence number of current log
CHECKPOINT_CHANGE#	NUMBER	SCN at last checkpoint
CHECKPOINT_TIME	VARCHAR2	Time of last checkpoint

V\$TIMER**The current time in hundredths of seconds.**

HSECS	NUMBER	Time in hundredths of a second

V\$TRANSACTION**Information about transactions.**

ADDR	RAW(4)	Address of transaction state object
SCNBASE	NUMBER	First part of the system change number; invalid if inactive
SCNWRAP	NUMBER	Second part of the system change number; invalid if inactive
XIDUSN	NUMBER	Rollback (undo) segment number; invalid if inactive
XIDSLOT	NUMBER	Slot number; invalid if inactive
XIDSQN	NUMBER	Sequence number; invalid if inactive
UBADBA	NUMBER	Database block address for rollback data; invalid if inactive
UBASQN	NUMBER	Sequence number for rollback data; invalid if inactive
UBAREC	NUMBER	Record number for rollback data; invalid if inactive

V\$TYPE_SIZE

Sizes of various database components for use in estimating data block capacity.

COMPONENT	VARCHAR2	Component name, such as segment or buffer header
TYPE	VARCHAR2	Component type
DESCRIPTION	VARCHAR2	Description of component
SIZE	NUMBER	Size of component

V\$VERSION

Version numbers of core library components in the ORACLE Server. There is one row for each component.

BANNER	VARCHAR2	Component name and version number

V\$WAITSTAT

Block contention statistics. This table is only updated when timed statistics are enabled.

CLASS	VARCHAR2	Class of block subject to contention
COUNT	NUMBER	Number of waits by this OPERATION for this CLASS of block
TIME	NUMBER	Sum of all wait times for all the waits by this OPERATION for this CLASS of block

C NATIONAL LANGUAGE SUPPORT

This appendix describes the database administrator's control of features that enable ORACLE applications to operate with multiple languages using conventions specified by the application user. Topics include:

- specifying language-dependent behavior for a session
- specifying default language-dependent behavior
- overriding the standard defaults
- data and number formats

Specifying Character Sets

Once the database is created, the database character set cannot be changed without re-creating the database. Hence, it is important to consider carefully which character set to use. The database character set should always be a superset of the operating system's native character set. The character sets used by client applications that access the database will usually determine which superset is the best choice.

If all client applications use the same character set, then this is the normal choice for the database character set. When client applications use different character sets, the database character set should be a superset (or equivalent) of all the client character sets. This will ensure that every character is represented when converting from a client character set to the database character set.

Character Set Conversion

Replacement characters may be defined for specific characters as part of a character set definition. Where a specific replacement character is not defined, a default replacement character is used. To avoid the use of replacement characters when converting from client to database character set, the latter should be a superset (or equivalent) of all the client character sets.

Specifying Language-Dependent Behavior for a Session

This section discusses the NLS parameters that specify language-dependent operation of applications.

NLS_LANG Parameter The NLS_LANG parameter has three components (language, territory, and charset) in the form:

NLS_LANG = language_territory.charset

Each component controls the operation of a sub-set of NLS features.

language specifies conventions such as the language used for ORACLE messages, day names, and month names. Each supported language has a unique name, for example American, French, or German. The language argument specifies default values for the territory and character set arguments, so either (or both) *territory* or *charset* can be omitted.

territory specifies conventions such as the default date format and decimal character used for numbers. Each supported *<territory>* has a unique name, for example America, France, Canada.

charset specifies the character set used by the client application (normally that of the user's terminal). Each supported character set has a unique acronym, for example US7ASCII, WEISO8859P1, WE8DEC, WE8EBCDIC500, JA16EUC.



Each language has a default character set associated with it. Default values for the languages available on your system are listed in your installation or user's guide.

The three arguments of NLS_LANG can be specified in any combination, for example:

AMERICAN_AMERICA.US7ASCII

FRENCH_FRANCE.ISO8859P1

FRENCH_CANADA.WE8DEC

JAPANESE_JAPAN.JA16EUC

Specifying NLS_LANG NLS_LANG is defined for each session with an environment variable or equivalent platform-specific mechanism. Different sessions connected to the same database can specify different values for their NLS_LANG parameter.



Each operating system handles environment variables differently, so refer to your installation or user's guide to determine how to set NLS_LANG on your system. For example, on VMS you could specify the value of NLS_LANG by entering the following line at the VMS prompt:

```
$ DEFINE NLS_LANG FRENCH_FRANCE.WE8DEC
```

Client/Server Architecture

NLS_LANG sets the NLS environment used by the database for both the Server session and for the client application. Using the one parameter ensures that the language environments of both database and client application are automatically the same.

Default Character Set



Default values for language and territory can be specified for an instance by defining the two initialization parameters NLS_LANGUAGE and NLS_TERRITORY when the instance is started. These two parameters are discussed in the next section. See your installation or user's guide for a list of the standard defaults on your system.

Overriding Language and Territory Specifications

The default values for language and territory can be overridden for a session by using the ALTER SESSION statement. For example:

```
ALTER SESSION SET NLS_LANGUAGE = FRENCH NLS_TERRITORY = FRANCE
```

This feature is used implicitly to determine the language environment of the database for each session. An ALTER SESSION statement is automatically executed when a session connects to a database to set the values of the database parameters NLS_LANGUAGE and NLS_TERRITORY to those specified by the *language* and *territory* arguments of NLS_LANG. If NLS_LANG is not defined, an ALTER SESSION statement is not executed.

When NLS_LANG is defined, the implicit ALTER SESSION is executed for all instances the session connects to, for both direct and indirect connections. If the values of NLS parameters are changed explicitly with ALTER SESSION during a session, the changes are propagated to all instances that user session is connected to.

Specifying Default Language-Dependent Behavior

This section describes NLS_LANGUAGE and NLS_TERRITORY, the initialization parameters that specify the default language-dependent behavior for a session.

NLS_LANGUAGE Parameter

NLS_LANGUAGE specifies the default conventions used for the following session characteristics:

- language used for Server messages
- language used for day and month names and their abbreviations (specified in the SQL functions TO_CHAR and TO_DATE)
- symbols used for equivalents of AM, PM, AD, and BC
- default sorting sequence used for character data when ORDER BY is specified (GROUP BY uses a binary sort, unless ORDER BY is specified)



The value specified for NLS_LANGUAGE in the initialization file is the default for all sessions in that instance. Refer to your installation or user's guide to check which language conventions are supported. For example, to specify the default session language as French, the parameter should be set as follows:

```
NLS_LANGUAGE = FRENCH
```

In this case, the Server message

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

will appear as:

```
ORA-00942: table ou vue n'existe pas
```



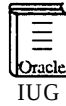
Messages used by the Server are stored in binary-format files that are placed in the ora_rdbms directory, or equivalent. (See your installation or user's guide for the location on your system.) Multiple versions of these files can exist, one for each supported language, using the filename convention:

```
<product_id><language_id>.MSB
```

For example, the file containing the Server messages in French is called ORAF.MSB, 'F' being the language abbreviation used for French.

Messages are stored in these files in one specific character set, depending on the particular machine and operating system. If this is different from the database character set, message text is automatically converted to the database character set. If necessary, it will be further

converted to the client character set, if it is different from the database character set. Hence, messages will be displayed correctly at the user's terminal, subject to the limitations of character set conversion.



The default value of NLS_LANGUAGE may be operating system specific. Refer to your installation or user's guide to check the default value. You can alter the NLS_LANGUAGE parameter by changing the value in the initialization file and then restarting the instance.

NLS_TERRITORY Parameter

NLS_TERRITORY specifies the conventions used for the following default date and numeric formatting characteristics::

- date format
- decimal character and group separator
- local currency symbol
- ISO currency symbol
- week start day

The value specified for NLS_TERRITORY in the initialization file is the default for the instance. For example, to specify the default as France, the parameter should be set as follows:

```
NLS_TERRITORY = FRANCE
```

In this case, numbers would be formatted using a comma as the decimal character.



The default value of NLS_TERRITORY may be operating system specific. Refer to your installation or user's guide to check the default value and to see which territory conventions are supported on your system. You can alter the NLS_TERRITORY parameter by changing the value in the initialization file and then restarting the instance.

Overriding the Standard Defaults

The NLS_LANGUAGE and NLS_TERRITORY parameters implicitly specify several aspects of language-dependent operation. Additional NLS parameters provide explicit control over these operations. The parameters listed below can be specified in the initialization file, or they can also be specified for each session with the ALTER SESSION command. (See the *ORACLE7 Server SQL Language Reference Manual* for a complete description of ALTER SESSION.)

The additional NLS parameters are:

Parameter	Description
NLS_DATE_FORMAT	default date format
NLS_DATE_LANGUAGE	default language used for dates
NLS_NUMERIC_CHARACTERS	decimal character and group separator
NLS_CURRENCY	local currency symbol
NLS_ISO_CURRENCY	ISO international currency symbol
NLS_SORT	character sort sequence

These parameters are described in the following sections.

NLS_DATE_FORMAT Parameter

This parameter specifies the default date format, overriding that defined implicitly by NLS_TERRITORY. Any format usable in the TO_DATE function can be specified with the NLS_DATE_FORMAT parameter. For example, to set the default date format to display Roman numerals for months, you would include the following line in your initialization file:

```
NLS_DATE_FORMAT = 'DD RM YY'
```

With such a default date format, the following SELECT statement would return the month using Roman numerals (assuming today's date is February 13, 1991):

```
SELECT TO_CHAR (SYSDATE) CURRDATE
       FROM DUAL;
```

```
CURRDATE
-----
13 II 91
```

Note: The applications you design may need to allow for a variable-length default date format.

You can alter the default value of NLS_DATE_FORMAT by changing its value in the initialization file and then restarting the instance, and you can alter the value during a session using an ALTER SESSION SET

NLS_DATE_FORMAT command. (See the *ORACLE7 Server SQL Language Reference Manual* for a description of the ALTER SESSION command.)

NLS_DATE_ LANGUAGE Parameter

This parameter specifies the language used for the spelling of day and month names by the functions TO_CHAR and TO_DATE, overriding that specified implicitly by NLS_LANGUAGE.

NLS_DATE_LANGUAGE has the same syntax as the NLS_LANGUAGE parameter, and all supported languages are valid values. For example, to specify the date language as French, the parameter should be set as follows:

```
NLS_DATE_LANGUAGE = FRENCH
```

In this case, the query

```
SELECT TO_CHAR (SYSDATE, 'Day:Dd Month yyyy')  
FROM DUAL;
```

would return:

Mercredi: 13 Février 1991

Month and day name abbreviations are also in the language specified, for example:

Me:13 Fév 1991

The default date format also uses the language-specific month name abbreviations. For example, if the default date format is DD-MON-YYYY, the above date would be inserted using:

```
INSERT INTO tablename VALUES ('13-Fév-1991');
```

The abbreviations for AM, PM, AD, and BC are also returned in the language specified by NLS_DATE_LANGUAGE. Note that numbers spelled using the TO_CHAR function always use English spellings, for example:

```
SELECT TO_CHAR (TO_DATE('27-Fév-91'), 'Day: ddspth Month')  
FROM DUAL;
```

would return

Mercredi: twenty-seventh Février

You can alter the default value of NLS_DATE_LANGUAGE by changing its value in the initialization file and then restarting the instance, and you can alter the value during a session using an ALTER SESSION SET NLS_DATE_LANGUAGE command. (See the *ORACLE7 Server SQL Language Reference Manual* for a description of the ALTER SESSION command.)

NLS_NUMERIC_CHARACTERS Parameter

This parameter specifies the decimal character and grouping separator, overriding those defined implicitly by NLS_TERRITORY. The decimal character separates the integer and decimal parts of a number. The grouping separator is the character returned by the number format mask G. For example, to set the decimal character to a comma and the grouping separator to a period, the parameter should be set as follows:

```
NLS_NUMERIC_CHARACTERS = ', .'
```

Both characters are single-byte and must be different. Either can be a space.



Note: When the decimal character is not a period (.) or when a group separator is used, numbers appearing in SQL statements must be enclosed in quotes, since the comma is usually used to separate list items. For example:

```
INSERT INTO SIZES (ITEMID, WIDTH, QUANTITY)
VALUES (618, '45,5', TO_NUMBER ('1.234', '9G999'));
```

You can alter the default value of NLS_NUMERIC_CHARACTERS by changing its value in the initialization file and then restarting the instance, and you can alter its value during a session using an ALTER SESSION SET NLS_DATE_LANGUAGE command. (See the *ORACLE7 Server SQL Language Reference Manual* for a description of the ALTER SESSION command.)

NLS_CURRENCY Parameter

This parameter specifies the character string returned by the number format mask L, the local currency symbol, overriding that defined implicitly by NLS_TERRITORY. For example, to set the local currency symbol to 'Dfl' (including a space), the parameter should be set as follows:

```
NLS_CURRENCY = 'Dfl '
```

In this case, the query

```
SELECT TO_CHAR (TOTAL, 'L099G999D99') "TOTAL"
FROM ORDERS WHERE CUSTNO = 586
```

would return:

```
          TOTAL
-----
Dfl 12.673,49
```

You can alter the default value of NLS_CURRENCY by changing its value in the initialization file and then restarting the instance, and you can alter its value during a session using an ALTER SESSION SET

NLS_CURRENCY command. (See the *ORACLE7 Server SQL Language Reference Manual* for a description of the ALTER SESSION command.)

**NLS_ISO_CURRENCY
Parameter**

This parameter specifies the character string returned by the number format mask C, the ISO currency symbol, overriding that defined implicitly by NLS_TERRITORY.

Local currency symbols can be ambiguous; for example, a dollar sign (\$) can refer to US dollars or Australian dollars. ISO Specification 4217 1987-07-15 defines unique 'international' currency symbols for the currencies of specific territories (or countries).

For example, the ISO currency symbol for the US Dollar is USD, for the Australian Dollar AUD. To specify the ISO currency symbol, the corresponding territory name is used.

NLS_ISO_CURRENCY has the same syntax as the NLS_TERRITORY parameter, and all supported territories are valid values. For example, to specify the ISO currency symbol for France, the parameter should be set as follows:

```
NLS_ISO_CURRENCY = FRANCE
```

In this case, the query

```
SELECT TO_CHAR (TOTAL, 'C099G999D99') "TOTAL"
      FROM ORDERS WHERE CUSTNO = 586
```

would return:

```
          TOTAL
-----
FRF12.673,49
```

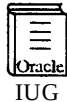
You can alter the default value of NLS_ISO_CURRENCY by changing its value in the initialization file and then restarting the instance, and you can alter its value during a session using an ALTER SESSION SET NLS_ISO_CURRENCY command. (See the *ORACLE7 Server SQL Language Reference Manual* for a description of the ALTER SESSION command.)

NLS_SORT Parameter This parameter specifies the type of sort used for character data, overriding that defined implicitly by NLS_LANGUAGE. The syntax of NLS_SORT is:

```
NLS_SORT = { BINARY | name }
```

BINARY specifies a binary sort and *name* specifies a particular linguistic sort sequence. For example, to specify the linguistic sort sequence called German, the parameter should be set as follows:

```
NLS_SORT = German
```



The name given to a linguistic sort sequence has no direct connection to language names. In most cases, however, each supported language will have an appropriate linguistic sort sequence defined that uses the same name. Refer to your installation or user's guide to check which linguistic sort sequences are supported.

You can alter the default value of NLS_SORT by changing its value in the initialization file and then restarting the instance, and you can alter its value during a session using an ALTER SESSION SET NLS_SORT command. (See the *ORACLE7 Server SQL Language Reference Manual* for a description of the ALTER SESSION command.)

Date and Number Formats

Several format masks are provided with the TO_CHAR, TO_DATE, and TO_NUMBER functions in order to format dates and numbers according to the relevant conventions.

Note: The TO_NUMBER function also accepts a format mask.

Date Formats

A format element RM (Roman Month) returns a month as a Roman numeral. Both uppercase and lowercase can be specified, using RM and rm respectively. For example, for the date 7 Sep 1992, 'DD-rm-YY' will return '07-ix-92' and 'DD-RM-YY' will return '07-IX-92'.

Note that the MON and DY format masks explicitly support month and day abbreviations that may not be three characters in length. For example, the abbreviations 'Lu' and 'Ma' can be specified for the French 'Lundi' and 'Mardi', respectively.

Week and Day Number Conventions

The week numbers returned by the WW format mask are calculated according to the algorithm $\text{int}((\text{day-jan1})/7)$. This week number algorithm does not follow the ISO standard (2015, 1992-06-15).

To support the ISO standard, a format element IW is provided that returns the ISO week number. In addition, format elements, I IY IYY and IYYY, equivalent in behavior to the format elements Y YY YYY YYYY, return the year relating to the ISO week number.

In the ISO standard, the year relating to an ISO week number can be different from the calendar year. For example 1st Jan 1988 is in ISO week number 53 of 1987. A week always starts on a Monday and ends on a Sunday.

- If January 1 falls on a Friday, Saturday, or Sunday, then the week including January 1 is the last week of the previous year, because most of the days in the week belong to the previous year.
- If January 1 falls on a Monday, Tuesday, Wednesday, or Thursday, then the week is the first week of the new year, because most of the days in the week belong to the new year.

For example, January 1, 1991, is a Tuesday, so Monday, December 31, 1990, to Sunday, January 6, 1991, is week 1. Thus the ISO week number and year for December 31, 1990, is 1, 1991. To get the ISO week number, use the format mask 'IW' for the week number and one of the 'IY' formats for the year. See the *ORACLE7 Server SQL Language Reference Manual* for more information on using date format masks.

Number Formats

Several additional format elements are provided for formatting numbers:

- D (Decimal) returns the decimal character
- G (Group) returns the group separator
- L (Local currency) returns the local currency symbol
- C (international Currency) returns the international currency symbol
- RN (Roman Numeral) returns the number as its Roman numeral equivalent

For Roman numerals, both uppercase and lowercase can be specified, using RN and rn, respectively. The number to be converted must be an integer in the range 1 to 3999.

D

DATABASE LIMITS

This appendix lists the limits of values associated with various database functions and objects.



Limits exist on several levels in the database. There is usually a hard-coded limit in the database that cannot be exceeded. The value may be further restricted for any given operating system. You must refer to your installation or user's guide to determine the maximum value of such limits.

You can also set limits when you create a database. These limits are recorded in the control file and cannot be overridden during the life of the database. Next, you can set limits for the duration of an instance with initialization parameters. These parameters temporarily override the database limits of the control file.

For example, the maximum number of database files allowed by the ORACLE Server can be reduced for a particular database by specifying a lower value for the MAXDATAFILES option of the CREATE DATABASE command. This limit is then recorded in the control file and cannot be exceeded for the life of that database. You can alter the value of the initialization parameter DB_FILES to a value less than MAXDATAFILES for a particular instance. The maximum number of database files is then limited to the number specified by DB_FILES for the life of that instance.

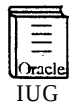
<i>Item</i>	<i>type</i>	<i>Limit</i>
blocks (ORACLE)	minimum in initial extent	2 blocks minimum (automatically enforced)
blocks (ORACLE)	maximum	2**32-1 (up to 4 terabytes, depending on block size)
characters	CHAR column index	255 characters maximum no absolute limit, but a function of block size LONG column 2 ³¹ -1 characters (2 gigabytes) maximum VARCHAR2 column 2000 characters maximum
columns	index (or cluster index)	16 columns maximum
	table	254 columns maximum
	expression list	254 columns maximum
	view definition	254 columns maximum
(LONG)	table	1 LONG column per table
constraints	CHECK (on columns)	unlimited
context area	size	no absolute limit (1024 is the minimum initial extent size)
control files	number of	one minimum; 2 or more strongly recommended on separate devices
control file	size	typically 50..200Kb, depending on database creation options; maximum is O/S-dependent
database file size	minimum	no absolute limit except for first file whose minimum size is 2 MB
	maximum	O/S dependent, typically 16 million ORACLE blocks

<i>Item</i>	<i>type</i>	<i>Limit</i>
database files	system	1022 or value of DB_FILES in INIT.ORA, or limited by value of MAXDATAFILES in CREATE DATABASE. Less on some operating systems.
GROUP BY clause	maximum size	number of bytes limited to one ORACLE block, less O/S-dependent block overhead, less 2-bytes per group-by expression, less one of the following: 2 bytes plus size of each aggregate of a non-distinct value Example: COUNT(DISTINCT(X)) or two bytes plus size in bytes of the longest aggregate of a <i>distinct</i> value Example: COUNT(x)
indexes	table	no limit
instances	parallel server	O/S dependent, subject to ORACLE limit of 255
literals	character string number (+ or -)	255 characters 1.0×10^{-130} to 1.0×10^{125}
locks	transaction distributed	no limit O/S dependent
MAXEXTENTS		O/S dependent
nested queries		255 queries
NUMBER	maximum value	1.0×10^{125}
precision		up to 38 significant digits per numeric value

<i>Item</i>	<i>type</i>	<i>Limit</i>
redo log files	database	255 or value for LOG_FILES in INIT.ORA or by MAXLOGFILES in CREATE DATABASE. Ultimately, an operating system limit.
redo log file size	minimum	50 Kbytes
rollback segments	database	no limit
rows	table	no limit
SGA size	maximum	no limit
SQL statement length		64 K maximum length; particular tools may have lower limits
stored packages	size	SQL*FORMS may have limits on the size of stored procedures you can call. Consult your SQL*Forms documentation for details.
tablespaces	database	no limit
tables	cluster	32
	database	no limit
trigger cascade limit	maximum	32 larger values O/S-dependent

E

OPERATING SYSTEM SPECIFIC INFORMATION



This Guide occasionally refers to other ORACLE manuals that contain detailed information for using ORACLE on a specific operating system. These ORACLE manuals are often called *installation or user's guides*, although the exact name may vary on different operating systems. Throughout this guide, references to these manuals are marked with the icon shown in the left margin.

This appendix lists all the references in this guide to operating system specific ORACLE manuals, and lists the operating system (OS) dependent initialization parameters. If you are using ORACLE on multiple operating systems, this appendix can help you ensure that your applications are portable across these operating systems.

Operating system specific topics are listed alphabetically, with page numbers of sections that discuss these topics.

Topic	Page
Administrator's operating system account	1-4
Archiving	
initial mode	17-3
enabling automatic archiving	17-4
operating system specific steps	17-4
Auditing	
creation of audit trail views	13-4
directing records to operating system's audit trail	13-14
operating system auditing	13-2, 12-22
Authenticating users with the operating system	17-4
Backing up	18-8, 18-10 to 18-14
Character sets, language default	C-3
Clusters, estimating size	8-43
Control files	
default name	2-6
filename specification	6-3
maximum size	6-3, 2-11, D-2
Connect strings	4-9
Creating a database	
during installation	1-10
operating-specific steps	2-3
Data files	
creating and naming for installation	7-5
limit on number of	7-2
Database	
block size	A-14
database limits	D-1 to D-4
default directory	5-5
Database links,	
connect string	15-8
limit on number of connections per process	4-5
Dedicated server, requesting	4-9
Disk use, monitoring	22-3
Dispatcher processes	
connect strings	4-4
number depends on number of connections per process	4-5
Distributed transaction recovery for single process	
instances	15-29

Topic	Page
Extents, maximum number per table	8-17
Files	
created initially	7-5
number allowed in a database	7-2
specifying names	2-6
Hardware, evaluating	1-9
Indexes, space required	8-35
Installing products	1-8
Instance	
checking the instance identifier	4-11
startup	3-5
INTERNAL	
and non-secure connections	1-5
connecting as	1-4,2-9
password for	1-4
protection for	10-7
Latch contention threshold	23-12
Listener process, address of	4-4
MAXLOGFILES, MAXLOGMEMBERS	5-4
Monitors, running MONITOR.SQL	B-73
Migration from Version 6 to ORACLE7	2-3
National Language Support (NLS)	
language-specific defaults	C-C
linguistic sort sequences supported	A-36, C-11
location of error message files	C-5
NLS_LANG environment variable	C-4
NLS_LANGUAGE	A-34, A-34, C-5, C-6
NLS_SORT	A-36
NLS_TERRITORY	A-36, C-6
Parallel Server	
multi-threaded server connect strings	4-4
support for	3-5
Parallel writes, overhead	22-4

Topic	Page
Parameter files	
case-sensitive filenames	A-2
creating and editing	3-14
default filename	3-3
format	A-8
global cache parameter values	A-5
location	2-5, A-1
operating system dependent parameters	A-5
Recovery	
suggested log file names in SQL*DBA	19-13
Redo log files	
specifying the destination search path	19-13
default location	5-5
default size	5-3
limited by LOG_FILES initialization parameter	5-4
REFSNAP utility	16-2
Resource costs, setting	11-20
Roles	
OSOPER and OSDBA, names and functionality	1-5
using the operating system for security	12-22, 10-3, 10-7
security	10-3
SQL scripts	
CATNOAUD.SQL, location	13-5
DBMSSNAP.SQL, name and location	16-2
file names and locations	2-16, B-2
to run initially	2-13, 2-16
UTLCHAIN.SQL, name and location	8-66, 22-7
UTLEXCPT.SQL, location	8-74
Storage parameters, defaults	8-6
SQL*DBA	
installing	8-6
setup	3-14
starting	2-9
suggested log file names, example	19-13
Tables, estimating size	
clustered	8-43
non-clustered	8-18
Tablespaces, default storage parameters	8-6
Trace files	4-12
Tuning	21-4

Topic	Page
Initialization parameters	
BACKGROUND_DUMP_DEST	A-9
COMMIT_POINT_STRENGTH	15-15, A-10
CONTROL_FILES	2-6, A-11
DB_BLOCK_SIZE	A-14
DB_FILES	A-15
DB_FILE_MULTIBLOCK_READ_COUNT	A-15
DB_FILE_SIMULTANEOUS_WRITES	A-16
DISTRIBUTED_TRANSACTIONS	A-19
INIT_SQL_FILES	2-13, A-22
INSTANCE_NUMBER	A-22
LOG_ARCHIVE_BUFFER_SIZE	A-24
LOG_ARCHIVE_BUFFERS	A-25
LOG_ARCHIVE_DEST	17-12, A-25
LOG_ARCHIVE_FORMAT	17-12, A-26
LOG_BUFFER	A-27
LOG_CHECKPOINT_INTERVAL	A-27
LOG_SMALL_ENTRY_MAX_SIZE	23-10, A-29
MTS_MAX_DISPATCHERS (maximum value)	23-6, A-31
MTS_MAX_SERVERS (maximum value)	23-8, A-32
MTS_SERVERS	A-32
NLS_LANGUAGE	A-34, A-34, C-5, C-6
NLS_TERRITORY	A-36, C-6
OPEN_CURSORS (value, maximum value)	21-6, A-37
OS_AUTHENT_PREFIX (value, case sensitivity)	7-5, A-38
PROCESSES	A-39
SHARED_POOL_SIZE (value, maximum value)	21-10, 21-15, A-44
SMALL_TABLE_THRESHOLD	A-44
SORT_AREA_SIZE	24-2, A-45
SORT_SPACEMAP_SIZE	A-46
TEMPORARY_TABLE_LOCKS	A-47
TRANSACTIONS_PER_ROLLBACK_SEGMENT	A-48
USER_DUMP_DEST	A-49

INDEX

A

Abbreviations

AM/PM C-8

BC/AD C-8

K (kilobyte) 8-6

M (megabyte) 8-6

Abort

shutting down an instance 3-11

Access, data

managing 12-1

system privileges 12-2

Access, data dictionary

DBA views B-5

Access, database

controlling 11-1

database administrator account 1-3

granting privileges 12-16

restricting 3-4

revoking privileges 12-18

Access, object

data dictionary views B-5

dynamic performance tables B-73

granting privileges 12-17

privilege types 12-9

revoking privileges 12-19

ACCESSIBLE_COLUMNS view B-15

Accounts, operating system

database administrator 1-3

role identification and 12-24

Accounts, user

SYS 1-6

SYSTEM 1-6

Add Data Files to Tablespace dialog 7-7

Add Online Redo Log Group dialog box 5-4

Add Online Redo Log Member dialog 5-6

Adding

control files 6-4

dispatcher processes 23-6

ADMIN OPTION 12-16

revoking 12-19

Administering ORACLE databases 1-1

AFTER ROW triggers

snapshot log creation and 16-12

AFTER triggers

auditing and 13-21

ALERT file A-9

location 4-12

session high water mark in 11-6

size 4-12

using 4-11

when written 4-12

ALL_CATALOG view B-5

ALL_COL_COMMENTS view B-5

ALL_COL_PRIVS view B-5

ALL_COL_PRIVS_MADE view B-5

ALL_COL_PRIVS_RECD view B-6

ALL_CONS_COLUMNS view B-6

ALL_CONSTRAINTS view B-6

ALL_DB_LINKS view B-7

ALL_DEF_AUDIT_OPTS view B-7

ALL_DEPENDENCIES view B-8

ALL_ERRORS view B-8

ALL_IND_COLUMNS view B-10

ALL_INDEXES view B-9

filling with data 8-63

ALL_LABELS view B-10
 ALL_MOUNTED_DBS view B-10
 ALL_OBJECTS view B-11
 ALL_SEQUENCES view B-11
 ALL_SNAPSHOTS view B-12
 ALL_SOURCE view B-13
 ALL_SYNONYMS view B-13
 ALL_TAB_COLUMNS view B-15
 ACCESSIBLE_COLUMNS synonym B-15
 filling with data 8-63
 ALL_TAB_COMMENTS view B-15
 ALL_TAB_PRIVS view B-16
 ALL_TAB_PRIVS_MADE view B-16
 ALL_TAB_PRIVS_RECD view B-16
 ALL_TABLES view B-14
 filling with data 8-63
 ALL_TRIGGER_COLS view B-17
 ALL_TRIGGERS view B-17
 ALL_USERS view B-17
 ALL_VIEWS view B-18
 Allocation
 archival buffer A-25
 extents 8-25
 extents for clusters 8-52
 minimizing extents for rollback segments 9-14
 rollback segments A-41
 Allocation, of memory
 See Memory allocation
 Alphanumeric data
 See Datatypes, character
 ALTER CLUSTER command
 ALLOCATE EXTENT option 8-52
 MAXTRANS option 8-8
 using for hash clusters 8-60
 using for index clusters 8-52
 ALTER DATABASE command 3-7
 ADD LOG MEMBER option 5-6
 ADD LOGFILE A-47
 ADD LOGFILE option 5-5
 ARCHIVELOG option 17-3
 BACKUP CONTROLFILE TO TRACE
 option 18-15
 CREATE DATAFILE option 19-8
 CREATE DATAFILE parameter 19-8
 database name A-17
 DATAFILE..OFFLINE DROP option 7-12
 ALTER DATABASE command *continued*
 DROP LOGFILE MEMBER option 5-12
 DROP LOGFILE option 5-9
 ENABLE THREAD A-47
 EXCLUSIVE option 3-8
 MOUNT option 3-8
 NOARCHIVELOG option 17-3
 NORESETLOGS option 19-26
 OPEN NORESETLOGS option 19-20
 OPEN option 3-9
 PARALLEL option 3-8
 RECOVER clause 19-12
 RENAME FILE option 5-8,7-14
 RENAME_GLOBAL_NAME option 15-5
 RESETLOGS option 19-26
 THREAD A-47
 ALTER FUNCTION command
 COMPILE option 8-77
 ALTER INDEX command 8-39
 MAXTRANS option 8-8
 ALTER PACKAGE command
 COMPILE option 8-77
 ALTER PROCEDURE command
 COMPILE option 8-77
 ALTER PROFILE command 11-18
 COMPOSITE_LIMIT option 11-18
 Alter Profile dialog 11-17
 ALTER RESOURCE COST command 11-19
 ALTER ROLE command
 changing authorization method 12-14
 Alter Role dialog 12-13
 ALTER ROLLBACK SEGMENT
 ONLINE option 9-11
 ALTER ROLLBACK SEGMENT command 9-9
 OFFLINE option 9-12
 ONLINE option 9-10
 PUBLIC option 9-10
 STORAGE clause 9-9
 Alter Rollback Segment Storage dialog 9-9
 ALTER SEQUENCE command 8-30
 ALTER SESSION command
 ADVISE option 15-18
 SET GLOBAL_NAMES option 15-6
 SET NLS_CURRENCY option C-9 to C-10
 SET NLS_DATE_FORMAT option C-7
 SET NLS_LANGUAGE option C-4

- ALTER SESSION command *continued*
 - SET NLS_NUMERIC_CHARACTERS option C-9
 - SET NLS_TERRITORY option C-4
 - SET SQL_TRACE parameter 4-11
- ALTER SNAPSHOT command 16-5
 - REFRESH clause 16-5
- ALTER SNAPSHOT LOG command 16-13
- ALTER SYSTEM command
 - ARCHIVE LOG ALL option 17-7
 - ARCHIVE LOG START option 17-5
 - ARCHIVE LOG START option, specifying destination 17-12
 - ARCHIVE LOG STOP option 17-6
 - CHECKPOINT option 5-15
 - DISABLE DISTRIBUTED RECOVERY option 15-28
 - ENABLE DISTRIBUTED RECOVERY option 15-28
 - ENABLE RESTRICTED SESSION option 3-9
 - KILL SESSION option 4-14
 - SET GLOBAL_NAMES option 15-6
 - SET LICENSE_MAX_SESSIONS option 11-4
 - SET LICENSE_MAX_USERS option 11-5
 - SET LICENSE_SESSIONS_WARNING option 11-4
 - SET MTS_DISPATCHERS option 4-7
 - SET MTS_SERVERS option 4-6
 - SET NLS_LANGUAGE option C-4
 - SET RESOURCE_LIMIT option 11-22
 - SWITCH LOGFILE option 5-15
- ALTER TABLE command 8-24
 - ALLOCATE EXTENT option 8-25
 - DISABLE ALL TRIGGERS option 8-69
 - DISABLE integrity constraint option 8-73
 - DROP integrity constraint option 8-73
 - ENABLE ALL TRIGGERS option 8-69
 - ENABLE integrity constraint option 8-72
 - MAXTRANS option 8-8
- ALTER TABLESPACE command 7-7
 - ADD DATAFILE parameter 7-7
 - BEGIN BACKUP option 18-9,18-13
 - DEFAULT STORAGE option 7-8
 - END BACKUP option 18-10, 18-13
 - OFFLINE option 7-11, 18-14
 - ONLINE option 7-9, 18-14
- ALTER TABLESPACE command *continued*
 - RENAME DATA FILE option 7-13
- ALTER TRIGGER command
 - COMPILE option 8-77
 - DISABLE option 8-69
 - ENABLE option 8-69
- ALTER USER command 11-12
 - changing passwords with 11-12
- Alter User dialog 11-11
- ALTER USER privilege 11-14
- ALTER VIEW command
 - COMPILE option 8-77
- Altering
 - cluster indexes 8-52
 - clustered tables 8-52
 - clusters 8-51
 - database status 3-7
 - hash clusters 8-60
 - indexes 8-39
 - public rollback segments 9-10
 - rollback segment storage parameters 9-9
 - sequences 8-30
 - snapshot log storage parameters 16-13
 - storage parameters 8-24
 - tables 8-23 to 8-24
 - tablespace storage 7-8
 - users 11-11
- AM/PM
 - language of C-8
- ANALYZE command 22-7
 - CASCADE option 8-66
 - COMPUTE STATISTICS option 8-64
 - ESTIMATE STATISTICS SAMPLE option 8-64
 - LIST CHAINED ROWS option 8-66
 - number of cluster keys 8-49
 - recursive SQL and 8-65
 - shared SQL and 8-65
 - STATISTICS option 8-62
 - VALIDATE STRUCTURE option 8-66
- Analyzing objects 8-62
 - privileges 8-67
- Application administrator
 - See Database administrator
- Application design 1-ii
- Application developers
 - privileges for 10-8

- Application developers *continued*
 - responsibilities of 1-3
 - roles for 10-9
- Application development
 - distributed databases and 14-2
 - security for 10-8
- Applications
 - location in distributed processing 14-2
- ARCHIVE LOG command
 - destination file A-25
 - LIST parameter 5-9
 - START and STOP A-26
- Archived redo log
 - applying during recovery 19-12, 19-14
 - archiving modes 17-3
 - automatic vs. manual A-26
 - automatic, disabling archiving 17-6
 - buffer A-24 to A-25
 - default file name format 17-11
 - deleting after recovery 19-9
 - enabling automatic archiving 17-4
 - errors during recovery 19-15
 - file name format 17-10
 - listing status 17-9
 - location during recovery 19-12
 - loss of 19-34
 - preserving or resetting log sequence number 19-26
 - restoring to disk 19-8
 - setting archive destination 17-12
 - specifying archive destination 17-10
 - storage device A-25 to A-26
 - tuning A-24 to A-25
 - See also* Automatic archiving
- ARCHIVELOG mode
 - archiving destination A-25 to A-26
 - automatic archiving A-26
 - data file loss in 19-28
 - distributed database backups 18-4
 - enabling 17-3
 - setting at database creation 17-3
 - strategies for backups in 18-5
 - taking data files offline and online in 7-12
 - See also* Archiving
- Archiving
 - advantages 17-2
- Archiving *continued*
 - automatic, disabling 17-6
 - automatic, disabling after startup 17-6
 - automatic, disabling at startup 17-6
 - automatic, enabling 17-4
 - changing mode 17-3
 - default file name format 17-11
 - disadvantages 17-2
 - enabling and disabling 17-3
 - enabling and disabling at creation 17-3
 - enabling automatic 17-5
 - increasing speed of 17-9
 - manually 17-6
 - minimizing impact on system
 - performance 17-8
 - privileges for manual archiving 17-8
 - privileges to disable 17-6
 - privileges to enable 17-5
 - privileges to view status 17-10
 - specifying destination 17-10
 - tuning 17-8
 - viewing information on 17-9
- Assigning
 - rollback segments to transactions 22-11
- Audience
 - of this Guide 1-ii
- AUDIT command 13-10
 - schema objects 13-11
 - statement auditing 13-11
 - system privileges 13-11
- Audit trail
 - archiving 13-15
 - controlling size of 13-14
 - creating and deleting 13-4
 - deleting views 13-5
 - interpreting 13-17
 - maximum size of 13-15
 - parameter A-8
 - protecting integrity of 13-17
 - purging records from 13-15
 - recording changes to 13-17
 - records in 13-5
 - reducing size of 13-16
 - table that holds 13-2
 - views on 13-4
- AUDIT_ACTIONS table B-18

AUDIT_TRAIL parameter A-8

setting 13-14

Auditing

audit option levels 13-5

audit trail records 13-5

changes to the audit trail 13-17

default options 13-12

disabling 13-14

disabling audit options 13-12

disabling default options 13-13

disabling options 13-13

disabling options vs. auditing 13-12

distributed databases and 15-3

enabling 13-14

enabling options 13-10

enabling options vs. auditing 13-10

guidelines 13-2

historical information 13-3

keeping information manageable 13-2

managing 13-1

managing audit trail size 13-14

managing the audit trail 13-4

policies for 10-10

privilege 13-8

privileges required for object 13-12

privileges required for system 13-11

protecting the audit trail 13-17

schema object 13-9, 13-11

session level 13-7

shortcuts for object 13-10

shortcuts for system 13-7

statement 13-11

statement level 13-5

suspicious activity 13-3

triggers and 13-20

use of system privileges 13-11

using the database 13-2

viewing active object options 13-19

viewing active privilege options 13-19

viewing active statement options 13-18

viewing default object options 13-19

views 13-4

See also AUDIT command

See also Audit trail

Authentication

changing 11-12

Authentication *continued*

database managed 11-7

multi-threaded server and 11-7

operating system managed 11-6

password policy 10-4

specifying when creating a user 11-9

user, overview 10-2

users 11-6

Authorization

changing for roles 12-13

omitting for roles 12-13

operating system role management and 12-12

roles 12-11

roles and the multi-headed server 12-12

AUTO_MOUNTING parameter A-8

Automatic archiving

archive log destination 17-4

Availability

data files 7-12

tablespaces 7-9

B

Background processes

enabling the checkpoint process (CKPT) A-9

specifying number of A-39

specifying trace files A-9

tuning the log writer process (LGWR) A-9

See also ORACLE processes

BACKGROUND_DUMP_DEST

parameter 4-12, A-9

ALERT file A-9

Backups

after creating new databases 1-10

after structural changes to database 18-2

ARCHIVELOG mode in 18-5

before database creation 2-4

checking data file backup status 18-11

control files 18-15

crashes during 18-13

creating a strategy 18-4

distributed databases 18-4

effects of archiving on 17-2

Export utility and 18-3

failures during 18-17

frequency of 18-2

- Backups *continued*
 - guidelines for 18-2
 - keeping 18-3
 - listing files needed 18-6
 - marking end 18-10
 - NOARCHIVELOG mode in 18-5
 - often-used tablespaces 18-3
 - performing full backups 18-7
 - performing partial backups 18-8
 - planning for distributed databases 15-3
 - privileges 18-12, 18-14
 - privileges for control files 18-17
 - procedure for data files 18-9,18-13
 - procedure for tablespaces 18-9,18-13
 - procedures for 18-6
 - tablespaces 18-12
 - test strategies 18-2
 - why plan before database creation 18-2
- Base table
 - snapshots 16-4
- BC/AD
 - language of C-8
- Begin Manual Archive dialog box 17-7
- Begin Online Tablespace Backup dialog 18-9
- Bind variables
 - and shared SQL areas 21-11 to 21-12
- Block
 - redo log A-24, A-27
 - size A-14
- Block header
 - calculating size for tables 8-18
- Branch blocks
 - space required 8-37
- Bringing online
 - tablespaces 7-9
- Buffer cache 20-5
 - database A-12
 - memory allocation 21-18, 21-21
 - reducing cache misses 21-18
 - tuning 21-17
- Buffer, archival A-24 to A-25
- Buffers
 - determining number to add 21-20
 - effect on performance A-13
 - number of A-13
 - when to reduce number of 21-21

- Buffers, database
 - number of A-12
- Bug fixes 1-11
- BUSY column
 - of V\$DISPATCHER table 23-4

C

- Cache
 - database blocks A-12
 - dictionary B-87
 - row cache A-41
 - sequence cache A-42 to A-43
 - Statistics A-27
- CASCADE option
 - integrity constraints 8-53
 - when dropping unique or primary keys 8-73
- Cascading revokes 12-20
- CAT synonym for USER_CATALOG B-18
- CATALOG. SQL 2-14, B-2
 - creating V\$ views B-72
 - including in INIT_SQL_FILES A-22
- CATALOG6.SQL 2-16, B-4
- CATAUDIT.SQL 2-14, B-2
 - running 13-4
- CATBLOCK.SQL 2-16, B-2
 - running 4-10
- CATDBSYN.SQL 2-16, B-2
- CATDDB.SQL B-2
- CATEXP.SQL 2-14, B-2
- CATEXP6.SQL 2-16, B-4
- CATLDR.SQL 2-14, B-2
- CATNOAUD.SQL 2-16, B-2
 - running 13-5
- CATNOPRC.SQL 2-16, B-2
- CATPARR.SQL 2-14, B-2
- CATPRC.SQL 2-14
- CATPROC.SQL 2-14, B-2
- CATRPC.SQL 2-14
- CATSNAP.SQL 2-14
 - running 16-2
- CATSTAT.SQL 2-16
 - running 8-66
- CATTRST.SQL 2-14
- CHAINED_ROWS table B-18
- Chaining rows 22-7

- Change-based recovery
 - procedure for 19-23
- Changing
 - passwords 11-12
- CHAR datatype 8-11
 - increasing column length 8-24
 - space use of 8-11
- Character sets
 - multi-byte characters in role names 12-11
 - multi-byte characters in role passwords 12-11
 - multi-byte, and user names 11-8
 - multi-byte, and user passwords 11-9
 - parameter file and 3-14
 - specifying when creating a database 2-2
 - supported by ORACLE 8-11
- CHECK constraint
 - See* Data integrity; Integrity constraint
- Checkpoint process (CKPT) A-9
 - behavior on checkpoints 24-7
 - enabling 24-7
 - starting 4-13
- CHECKPOINT_PROCESS parameter A-9
 - setting 4-13, 24-7
- Checkpoints
 - checkpoint interval A-27
 - choosing checkpoint frequency 24-7
 - CKPT process A-9
 - controlling 5-13
 - effect on recovery time performance 24-6
 - effect on runtime performance 24-6
 - “fast” checkpoint 5-15
 - forcing 5-15
 - log switches and 5-13
 - for redo log maintenance 24-7
 - reducing performance impact of 24-7
 - setting intervals 5-13
 - signaling DBWR to write 24-6
 - statistics A-27
 - tuning 24-6
- CLEANUP_ROLLBACK_ENTRIES
 - parameter A-10
- Client-server systems
 - development of applications in 14-2
 - location of applications in 14-2
 - managing 14-1
- CLU synonym for USER_CLUSTERS B-18
- Cluster keys
 - ANALYZE command and 8-49
 - columns for 8-42
- Clustered tables
 - See* Tables, clustered
- Clusters
 - allocating extents 8-52
 - altering 8-51
 - analyzing statistics 8-62
 - auditing 13-6
 - choosing data 8-41
 - columns for cluster key 8-42
 - creating 8-49
 - dropped tables and 8-25
 - dropping 8-52
 - estimating space 8-42 to 8-43
 - guidelines for managing 8-41
 - index creation 8-50
 - location 8-43
 - managing 8-41
 - privileges for controlling 12-10
 - privileges for creating 8-51
 - privileges for dropping 8-54
 - snapshots in 16-3
 - specifying PCTFREE for 8-3
 - storage parameters 8-9
 - truncating 8-67
 - validating structure 8-66
 - why estimate space 8-43
 - See also* Hash clusters
- CODE_PIECES view B-18
- CODE_SIZE view B-18
- Cold backups
 - See* Full backups
- COLS synonym for USER_TAB_COLUMNS B-18
- COLUMN_PRIVILEGES view B-18
- columns
 - displaying information about 8-82
 - granting privileges for selected 12-17
 - increasing length 8-24
 - INSERT privilege and 12-9
 - listing users granted to 12-28
 - revoking privileges from 12-19
 - revoking privileges on 12-19

- Command
 - See* individual command names
- Comments
 - in COMMIT statements 15-18
 - in parameter files A-6
 - on ORACLE documentation 1-ix
- COMMIT command
 - COMMENT parameter 15-18, 15-27
 - FORCE option 15-25 to 15-26
 - forcing 15-17
- Commit point site
 - setting commit point strength 15-15
- COMMIT_POINT_STRENGTH parameter
 - A-10
 - setting 15-15
- Communities
 - SQL*Net 15-3
- COMPATIBLE parameter A-11
- COMPATIBLE_NO_RECOVERY
 - parameter A-11
- Complete exports 18-18
- Complete recovery
 - procedures for 19-17
- Complete refreshes 16-5
- Composite limits 11-18
 - costs and 11-19
 - service units 11-18
- COMPUTE STATISTICS option 8-64
- Concurrency
 - ROW_LOCKING parameter A-42
 - sequences A-42
 - SERIALIZABLE parameter A-43
 - transactions A-48
 - transactions and A-48
 - user processes A-39
- Configure Multi-Threaded Dispatchers
 - dialog 4-7
- CONNECT
 - shortcut for auditing 13-7
- CONNECT INTERNAL
 - See* INTERNAL, connecting as
- CONNECT role 12-13
- Connect strings
 - and database links 15-7
- Connections
 - auditing 13-7
- Connections *continued*
 - changing hold time 15-27
 - dedicated servers 4-9
 - during shutdown 3-10
 - RECO 15-9
- Consistency
 - SERIALIZABLE parameter A-43
- consistent gets* statistic 21-17, 23-3, 24-6
 - calculating hit ratio 21-19, 21-22
- constraints
 - prohibited on snapshots 16-9
- Contention
 - block-level B-95
 - for disk access 22-2
 - for memory access 23-1
 - for free lists 24-5
 - for redo allocation latch 20-5, 23-13
 - for redo copy latches 20-5, 23-13
 - for rollback segments 23-2 to 23-3
 - statistics B-95
 - tuning 23-1
- Context areas
 - sort spacemap A-46
- Control files
 - backing up 18-7, 18-15
 - changing size 6-3
 - conflicts with data dictionary 6-6
 - creating 6-3
 - creating additional 6-4
 - creating initially 6-3
 - creating new 6-4
 - default name 2-6, 6-3
 - dropping 6-7
 - during incomplete recovery 19-23
 - finding filenames 18-7
 - guidelines for 6-2
 - location of 6-3
 - loss of 19-34 to 19-35
 - managing 6-1
 - mirroring 2-6
 - moving 6-4
 - names 6-2, A-11
 - number of 6-2
 - overwriting existing 2-6
 - privileges to backup 18-17
 - relocating 6-4

- Control files *continued*
 - renaming 6-4
 - requirement of one 6-3
 - size of 6-3
 - specifying A-11
 - specifying names before database creation 2-5
 - unavailable during startup 3-7
 - V\$BACKUP view and 18-11
 - why mirror 6-2
- CONTROL_FILES parameter A-11
 - overwriting existing control files 2-6
 - setting 6-2
 - setting before database creation 2-5, 6-3
- Conventions
 - used in this Guide 1-vi
- copying
 - tables 16-16
- Cost-based optimizer
 - data dictionary and 8-65
- costs
 - resource limits and 11-19
- COUNT column
 - of X\$KCB CBH table 21-21
 - of X\$KCB RBH table 21-18
- CPU_COUNT parameter
 - use 23-10
- CREATE CLUSTER command 8-50
 - for hash clusters 8-56
 - HASH IS option 8-57
 - HASHKEYS option 8-58
 - SIZE option 8-48, 8-57
- CREATE CONTROLFILE command 6-4
 - checking for inconsistencies 6-6
 - NORESETLOGS option 6-6
 - RESETLOGS option 6-5
- CREATE DATABASE command
 - CONTROLFILE REUSE option 6-3
 - database name A-17
 - example 2-10
 - initial SQL files A-22
 - MAXDATAFILES option 7-3, A-15
 - MAXLOGFILES option 5-3, A-28
 - MAXLOGMEMBERS option 5-4
 - setting the block size A-14
 - setting the log mode A-26
- CREATE DATABASE LINK command 15-7
 - CREATE DATABASE LINK command *continued*
 - CONNECT TO clause 15-8
 - PUBLIC option 15-8
- CREATE INDEX command 8-39
 - examples 24-4
 - NOSORT option 24-4
 - ON CLUSTER option 8-50
 - performance A-45
 - temporary space required 8-38
- CREATE PROFILE command 11-16
 - COMPOSITE_LIMIT option 11-18
- Create Profile dialog 11-16
- CREATE ROLE command 12-11
 - IDENTIFIED BY option 12-11
 - IDENTIFIED EXTERNALLY option 12-12
- Create Role dialog 12-11
- CREATE ROLLBACK SEGMENT
 - command 9-7
- Create Rollback Segment dialog 9-7
- CREATE SCHEMA command 8-60
 - privileges required 8-61
- CREATE SEQUENCE command 8-29
- CREATE SNAPSHOT command 16-2
 - REFRESH clause 16-5
- CREATE SNAPSHOT LOG command 16-11
- CREATE SYNONYM command 8-31
- CREATE TABLE command 8-22
 - CLUSTER option 8-50
 - examples 22-5 to 22-6
 - STORAGE clause 22-5
 - TABLESPACE clause 22-5
- CREATE TABLESPACE command 7-6
 - data filenames in 7-6
 - DATAFILE clause 22-5 to 22-6
 - examples 22-5 to 22-6
 - TABLESPACE clause 22-6
- Create Tablespace dialog 7-6
- CREATE USER command 11-8
 - IDENTIFIED BY option 11-9
 - IDENTIFIED EXTERNALLY option 11-9
- Create User dialog box 11-7
- CREATE VIEW command 8-26
 - OR REPLACE option 8-28
 - WITH CHECK OPTION 8-26

Creating

- audit trail 13-4
- cluster index 8-49
- clustered tables 8-49
- clusters 8-49
- control files 6-3
- data dictionary 2-13
- data files 7-5, 7-7
- database 2-1
- database links 15-7
- database, archiving mode 17-3
- database, backing up the new database 2-11
- database, during installation 2-3
- database, executing CREATE DATABASE 2-9
- database, maximum number of data files 7-3
- database, migration from different versions 2-3
- database, preparing to 2-2
- database, prerequisites for 2-3
- database, problems encountered while 2-11
- database, setting block size A-14
- database, specifying maximum number of data files A-15
- databases 1-10
- databases, steps 2-4
- hash clustered tables 8-56
- hash clusters 8-56
- indexes 8-38
- indexes, explicitly 8-39
- multiple objects 8-60
- online redo log groups 5-4
- parameter file 2-4
- profiles 11-16
- redo log members 5-6
- roles 12-11
- rollback segments 9-7, 23-3
- rollback segments, specifying storage parameters 9-8
- sequences 8-29
- snapshot logs 16-11
- snapshots 16-2
- snapshots and snapshot logs 16-10
- synonyms 8-31
- tables 8-22
- tablespaces 7-5
- tablespaces, rollback segments required 7-5

Creating *continued*

- users 11-7
- views 8-26

Cumulative exports 18-18

- view of last export B-30

Currency symbol

- default C-6

CURSOR_SPACE_FOR_TIME parameter A-12

- setting 21-13

Cursors

- row cache A-41

D

Data

- distributed 15-2
- security of 10-3

Data blocks

- “striping” 22-5
- altering size of 2-7
- factors affecting size of 2-6
- managing space use of 8-2
- operating system blocks vs. 2-7
- reading multiple A-15
- size of 2-6, A-14

Data dictionary

- adding to 2-16
- cache B-87
- creating 2-13
- DBA views B-5
- descriptions of tables within B-1
- See also* DICTIONARY view
- dropped tables and 8-25
- dynamic performance tables B-72
- gathering statistics on 8-65
- list of SQL scripts to create B-2
- location of 2-13
- rollback segments A-41
- row cache A-41
- schema object views 8-81
- segments in the 8-79
- sequence cache A-42
- setting storage parameters of 8-79
- user views B-5
- V\$DISPATCHER view 4-7
- V\$QUEUE view 4-7

Data dictionary *continued*

- views and B-5
- when to change storage parameters 8-81
- See also* DICTIONARY view
- See also* Data dictionary cache

Data dictionary cache

- reducing cache misses 21-15
- tuning 21-14

Data files

- adding to a tablespace 7-7
- backing up offline 18-13
- backing up online 18-9
- batch writes A-16
- bringing online and offline 7-12
- checking associated tablespaces 7-19
- copying 7-16
- creating 7-5
- database administrator's access to 1-3
- default directory 7-7
- dropping 7-17
- dropping in NOARCHIVELOG mode 7-12
- fulling specifying file names 7-7
- identifying filenames 7-15
- location 7-4
- loss of 19-28
- managing 7-1
- maximum number of 7-3, A-15
- minimum number of 7-2
- MISSINGnnnn 6-6
- monitoring 7-19
- parameter A-15
- placement on disk 22-4
- policies for managing 7-2
- recovering without a backup 19-8
- recovery creation 19-8
- relocating 7-13
- renaming 7-13
- requirements for renaming 7-13
- reusing 7-7
- size of 7-3
- storing separately from redo log files 7-4
- unavailable when database is opened 3-7
- viewing backup status 18-11
- viewing general status of 7-20
- viewing which need recovery 19-6

Database

- administering a 1-1
- availability 3-7
- control files of 6-2
- See* Creating a database
- creating and opening 1-10
- exclusive mode 3-5
- global name in a distributed system 15-5
- initial SQL files A-22
- location in distributed environment 15-2
- mounting to an instance 3-7
- name 15-5, A-17
- number of redo log files A-28
- parallel mode 3-5
- planning 1-9
- restricting access to 3-9
- steps to create 2-4
- why use global names 15-2

Database administrator

- application administrator vs. 10-10
- data dictionary views B-5
- distributed database considerations 15-2
- initial priorities 1-8
- manual archiving A-26
- network issues 15-2
- operating system account 1-3
- responsibilities of 1-2
- security and privileges 1-3
- security for 10-6
- security officer vs. 1-2, 10-2
- usernames 1-6
- using roles for security 10-7
- utilities 1-7

Database design

- implementing 1-10

Database limits D-1

Database links

- auditing 13-6
- creating 15-7
- dropping 15-9
- guidelines for managing 15-7
- limiting number of 15-10
- listing 15-10
- managing 15-7
- partial database names in 15-8
- partially specified 15-7

Database links *continued*

- private 15-8
 - privileges for controlling 12-10
 - privileges for creating 15-9
 - privileges for dropping 15-10
 - public 15-8
 - remote user privileges 15-9
 - security issues 15-12
 - specifying remote username 15-8
 - why create 15-2
- Database migration 2-3
- Database name
- changing 6-4
 - starting an instance and 3-2
- Database server hardware
- evaluating 1-8
- Database size, managing 7-1
- Database startup
- See* Starting an instance
- Database structure
- distributed database 1-9
 - logical 1-9
 - physical 1-9
- Database tuning 1-11
- Database user
- responsibilities of 1-3
- Database Writer process (DBWR)
- batch writes A-16
 - behavior on checkpoints 24-6
 - checkpoint A-27
- Databases
- checking the default instance 2-9
 - See also* CREATE DATABASE command
 - global name in a distributed system 2-5
 - names 2-5
 - production 10-8, 10-10
 - specifying control files 2-5
 - starting before database creation 2-9
 - test 10-8
 - tuning archiving for large 17-8
- DATAFILE clause
- examples 22-5
 - of CREATE TABLESPACE
 - command 22-5 to 22-6
- Datatypes
- character 8-11

Datatype *continued*

- space use of 8-11
 - summarized 8-14
 - See also* individual type names
- DATE datatype 8-12
- Dates
- default format C-6 to C-7
 - language of C-8
 - setting SYSDATE A-20
- Day
- format element C-8
 - language of names C-8
- db block gets* statistic 21-17, 23-3
- calculating hit ratio 21-19, 21-22
- DB_BLOCK_BUFFERS parameter A-12
- reducing buffer cache misses 21-18
 - removing unneeded buffers 21-21
 - setting before database creation 2-7
- DB_BLOCK_CHECKPOINT_BATCH
- parameter A-13
- DB_BLOCK_LRU_EXTENDED_STATISTICS
- parameter A-13
 - setting 21-18
- DB_BLOCK_LRU_STATISTICS
- parameter A-14
 - setting 21-21
- DB_BLOCK_SIZE parameter A-12, A-14
- database buffer cache size and 2-7
 - setting before creation 2-6
- DB_DOMAIN parameter 15-5, A-14
- setting 15-4
 - setting before database creation 2-5
- DB_FILE_MULTIBLOCK_READ_COUNT
- parameter A-15
- DB_FILE_SIMULTANEOUS_WRITES
- parameter A-16
- DB_FILES parameter A-15
- default 7-3
 - optional use of 7-3
- DB_MOUNT_MODE
- Trusted ORACLE parameter A-16
- DB_NAME parameter 15-4, A-17
- MTS_SERVICE and 4-4
 - setting 15-4
 - setting before database creation 2-5

DBA
 shortcut for auditing 13-7
 See also Database administrator
 DBA data dictionary objects B-5
 See also Database Administrator (DBA)
 DBA role 1-5, 12-13
 DBA-SYNONYMS.SQL
 See CATDBSYN.SQL
 DBA_2PC_NEIGHBORS view B-19
 DBA_2PC_PENDING view B-19
 DBA_AUDIT_EXISTS view B-20
 DBA_AUDIT_OBJECT view B-21
 DBA_AUDIT_SESSION view B-22
 DBA_AUDIT_STATEMENT view B-23
 DBA_AUDIT_TRAIL view B-24
 DBA_BLOCKERS view B-25
 DBA_CATALOG view B-25
 DBA_CLU_COLUMNS view B-26
 DBA_CLUSTERS view B-26
 DBA_COL_COMMENTS view B-26
 DBA_COL_PRIVS view B-27
 DBA_CONS_COLUMNS view B-27
 DBA_CONSTRAINTS view B-27
 DBA_DATA_FILES view B-28
 DBA_DB_LINKS view B-28
 DBA_DDL_LOCKS view B-28
 DBA_DEPENDENCIES view B-28
 DBA_DML_LOCKS view B-29
 DBA_ERRORS view B-29
 DBA_EXP_FILES view B-29
 DBA_EXP_OBJECTS view B-30
 DBA_EXP_VERSION view B-30
 DBA_EXTENTS view B-30
 DBA_FREE_SPACE view B-30
 DBA_INI_COLUMNS view B-32
 DBA_INDEXES view B-31
 filling with data 8-63
 DBA_LOCKS view B-32
 DBA_OBJ_AUDIT_OPTS view B-34
 DBA_OBJECT_SIZE view B-33
 DBA_OBJECTS view B-33
 DBA_PRIV_AUDIT_OPTS view B-34
 DBA_PROFILES view B-35
 DBA_ROLE_PRIVS view B-35
 DBA_ROLES view B-35
 DBA_ROLLBACK_SEGS view 9-16, B-35

DBA_SEGMENTS view B-36
 DBA_SEQUENCES view B-37
 DBA_SNAPSHOT_LOGS view B-38
 DBA_SNAPSHOTS view B-38
 DBA_SOURCE view B-39
 DBA_STMT_AUDIT_OPTS view B-39
 DBA_SYNONYMS view B-39
 DBA_SYS_PRIVS view B-39
 DBA_TAB_COLUMNS view B-41
 filling with data 8-63
 DBA_TAB_COMMENTS view B-42
 DBA_TAB_PRIVS view B-42
 DBA_TABLES view B-40
 filling with data 8-63
 DBA_TABLESPACES view 7-17, B-41
 DBA_TRIGGER_COLS view B-43
 DBA_TRIGGERS view B-42
 DBA_TS_QUOTAS view B-43
 DBA_USERS view B-43
 DBA_VIEWS view B-43
 DBA_WAITERS view B-44
 DBMS_ALERT_INFO view B-45
 DBMS_DDL.ANALYZE_OBJECT()
 running 8-65
 DBMS_LOCK_ALLOCATED table B-45
 DBMS_SNAPSHOT.PURGE_LOG
 procedure 16-14
 DBMS_SNAPSHOT.REFRESH() 16-7
 DBMS_SNAPSHOT.REFRESH_ALL()
 called automatically 16-2
 DBMS_UTILITY.ANALYZE_SCHEMA()
 running 8-65
 DBMSALRT.SQL 2-14, B-3
 DBMSDESC.SQL 2-14
 DBMSLOCK.SQL 2-14, B-3
 DBMSMAIL.SQL 2-14, B-3
 DBMSOTPT.SQL 2-14, B-3
 DBMSPIPE.SQL 2-14, B-3
 DBMSSNAP.SQL 2-14, B-3
 running 16-2
 DBMSSTDIX.SQL 2-14
 DBMSTRST.SQL B-3
 DBMSUTIL.SQL 2-14, B-3
 DBWR checkpoints statistic A-27
 Decimal character C-9
 default C-6, C-9

- Decimal character *continued*
 - when not a period (.) C-9
- Dedicated server processes
 - starting 4-2
 - trace files for 4-11
 - when to connect with 4-9
- Default
 - archive log filename format 17-11
 - audit options 13-12
 - audit options, disabling 13-13
 - control file name 2-6, 6-3
 - currency symbol C-6
 - data block size 2-6
 - date format C-6 to C-7
 - date language C-8
 - decimal character C-6, C-9
 - group separator C-6, C-9
 - INITRANS value 8-8
 - instance 2-9
 - maximum data files 7-3
 - MAXTRANS value 8-8
 - parameter file name 3-3
 - PCTFREE option 8-2
 - PCTUSED option 8-3
 - profile 11-17
 - role 11-13
 - sorting mechanism C-11
 - storage parameters 7-4, 8-5, 8-9
 - tablespace quota 11-10
 - temporary tablespace 11-10
 - user tablespaces 11-9
- Deleting
 - table statistics 8-62
 - See also* Dropping
- Dependencies
 - displaying 8-83
- Dependencies, object
 - circular references in remote objects 15-12
- DEPTREE view B-45
- Destination
 - archiving redo log files A-25
 - BACKGROUND_DUMP_DEST A-9
 - overriding default A-25
 - trace files for A-9
 - USER_DUMP_DEST A-49
- Developers, application
 - See* Application developers
- Devices
 - storage A-46
 - See also* Disk drives
- DICT synonym for DICTIONARY B-45
- DICT_COLUMNS view B-46
- DICTIONARY view
 - DICT synonym B-45
 - list of Data Dictionary contents B-46
- DICTSTAT.SQL B-2
- Dirty reads
 - done dirt cheap D-1
- Disabling
 - archiving 17-3
 - audit options 13-12 to 13-13
 - auditing 13-14
 - automatic archiving 17-6
 - global naming 15-6
 - integrity constraints 8-71
 - integrity constraints, effects on indexes 8-35
 - recoverer process 15-28
 - resource limits 11-21
 - triggers 8-69
- Disconnections
 - auditing 13-7
- DISCRETE_TRANSACTIONS_ENABLED
 - parameter A-17
- Disk contention 22-2
- Disk drives
 - archiving destination A-25
 - See also* Devices
- Disk failure
 - See* Media failure
- Dispatcher processes (Dnnn)
 - adding more 23-6
 - calculating maximum number of 4-5
 - number to start 4-4
 - privileges to change number of 4-9
 - removing 4-7
 - setting the number of 4-7
 - spawning new 4-7
- Dispatcher processes (Dnnn)
 - service name 4-4
- Distributed database option 1-i

- Distributed databases
 - administering 15-1
 - application development in 14-2
 - backup strategies 15-3
 - changing global names 15-5
 - choosing global object names 15-3
 - circular view or synonym references 15-12
 - coordinated time-based recovery 19-4
 - copying tables 16-16
 - creating global names 15-4
 - creating global object names 15-4
 - database administrator's role in 15-2
 - enforcement of global names 15-6
 - guidelines for managing 15-2
 - limiting active connections per user 15-10
 - location of databases 15-2
 - location transparency in 15-11
 - managing global names 15-3
 - media recovery and snapshots 19-5
 - name resolution in 15-11
 - number of transactions 15-27
 - problems in 15-15
 - recovery in 19-3
 - recovery strategies 15-3
 - remote object security and 15-12
 - running in ARCHIVELOG mode 17-2
 - running in NOARCHIVELOG mode 17-2
 - security issues 15-3, 15-12
 - starting a remote instance 3-6
 - taking backups 18-4
 - testing features 15-27
 - viewing global names 15-6
- Distributed processing
 - connecting between versions 14-3
 - installing network drivers 1-8
 - parameter file location in 3-14
 - See also* Client-server systems
- Distributed queries
 - enabling/disabling A-37
 - links for A-37
- Distributed transactions
 - failure during 15-15 to 15-16
 - forcing to fail 15-27
 - hold time 15-27
 - limiting number of 15-14, 15-27, A-19
 - locked resources in 15-16
- Distributed transactions *continued*
 - manually overriding in-doubt 15-17
 - recovery in single-process systems 15-29
 - resolution of in-doubt 15-14
 - setting advice for 15-18
- DISTRIBTED_LOCK_TIMEOUT
 - parameter A-18
 - controlling time-outs with 15-16
- DISTRIBUTED_RECOVERY_CONNECTION_
 - HOLD_TIME parameter 15-9, A-18
 - setting 15-27
- DISTRIBUTED_TRANSACTIONS
 - parameter A-19
 - recovery process and 15-14
 - setting 15-14, 15-27
 - when to alter 15-14
- Distributing I/O 2-12, 20-5, 22-4 to 22-6
- DIUTIL.SQL 2-14
- DML_LOCKS parameter A-19
- Domain
 - network 15-4
- DROP CLUSTER command 8-53, 8-60
 - CASCADE CONSTRAINTS option 8-53
 - INCLUDING TABLES option 8-53
- DROP DATABASE LINK command 15-9
- DROP INDEX command 8-41
- Drop Online Redo Log dialog 5-9
- Drop Online Redo Log Member dialog 5-11
- DROP PROFILE command 11-21
- Drop Profile dialog 11-20
- DROP ROLE command 12-15
- Drop Role dialog 12-15
- DROP ROLLBACK SEGMENT
 - command 9-15
- Drop Rollback Segment dialog 9-15
- DROP SYNONYM command 8-31
- DROP TABLE command 8-25
 - CASCADE CONSTRAINTS option 8-25
 - for clustered tables 8-53
- DROP TABLESPACE command 7-18
 - INCLUDING CONTENTS option 7-18
- Drop Tablespace dialog box 7-18
- DROP USER command 11-15
- Drop User dialog 11-14
- DROP USER privilege 11-15
- DROPCAT6.SQL B-4

- Dropping
 - audit trail 13-4
 - cluster indexes 8-52
 - clusters 8-52
 - control files 6-7
 - data files 7-17
 - database links 15-9
 - hash clusters 8-60
 - indexes 8-40
 - integrity constraints 8-73
 - integrity constraints, effects on indexes 8-35
 - online redo log groups 5-9
 - online redo log members 5-11
 - profiles 11-20
 - roles 12-15
 - rollback segments 9-10, 9-15
 - sequences 8-31
 - snapshot logs 16-15
 - snapshots 16-10
 - synonyms 8-31
 - tables 8-25
 - tablespaces 7-17
 - users 11-14
 - views 8-29
- Dump files
 - See Trace files
- Dynamic performance tables B-72
 - creating with CATALOG. SQL B-72
 - granting selective access B-73
 - public synonyms (V\$) B-72
 - using 4-10
 - views (V_\$) B-72

E

- Enabling
 - archiving 17-3
 - auditing 13-14
 - auditing options 13-10
 - Checkpoint process (CKPT) 24-7
 - recoverer process 15-28
 - resource limits 11-21
 - triggers 8-69
- Enabling integrity constraints 8-71
 - at creation 8-71
 - reporting exceptions 8-73

- Enabling integrity constraints *continued*
 - when violations exist 8-71
- Encryption
 - ORACLE passwords 11-7
- End Online Tablespace Backup dialog 18-10
- Enforcing
 - global names 15-6
- ENQUEUE_RESOURCES parameter A-20
- Enqueues
 - disabling A-19
- Enroll
 - database users 1-10
- Enter System Command dialog 5-8
- Environment variables
 - NLS_LANG C-3
- Error messages
 - ORA-02047 14-3
- ERROR_SIZE view B-46
- Errors
 - ALERT file and 4-11
 - archiving A-26
 - distributed transactions 15-15
 - during backup 18-13
 - during startup 3-7
 - indicating media failure 19-6
 - ORA-00028 4-14
 - ORA-00114 4-4
 - ORA-01090 3-10
 - ORA-01173 6-6
 - ORA-01176 6-6
 - ORA-01177 6-6
 - ORA-01591 15-17
 - ORA-02049 15-16
 - ORA-02050 15-15
 - ORA-02051 15-15
 - ORA-02054 15-15
 - ORA-1215 6-6
 - ORA-1216 6-6
 - ORA-1547 8-81
 - ORA-1628 through 1630 8-81
 - “Snapshot too old” 9-5
 - trace files and 4-11
 - when creating a database 2-11
 - when creating control file 6-6
 - while starting an instance 3-4
- ESTIMATE STATISTICS option 8-64

- Estimating size
 - hash clusters 8-55
 - indexes 8-35
 - tables 8-16
- Evaluating
 - hardware for the ORACLE server 1-8
- EVENT parameter A-20
- Examples
 - conventions in this Guide 1-vi
 - CREATE DATABASE command 2-10
 - of DATAFILE clause 22-5
 - manual transaction override 15-19
 - of NOSORT option 24-4
 - of CREATE INDEX command 24-4
 - of CREATE TABLE command 22-5 to 22-6
 - of CREATE TABLESPACE
 - command 22-5 to 22-6
 - of DATAFILE clause 22-6
 - of SET TRANSACTION command 22-11
 - of STORAGE clause 22-5
 - of table “striping” 22-5
 - of TABLESPACE clause 22-5 to 22-6
- Exceptions
 - integrity constraints 8-73
- EXCEPTIONS table B-46
- Exclamation point icon
 - use 1-vii
- Exclusive mode
 - killing remaining user sessions 4-13
 - of the database 3-5
 - rollback segments and 9-3
- EXP_FULL_DATABASE role 12-13
- Export files
 - view of last export B-29
- Export utility 1-7
 - backups and 18-3
 - complete exports 18-18
 - cumulative exports 18-18
 - incremental exports 18-18, B-30
 - modes for export in 18-18
 - read consistency and 18-17
 - restricted mode and 3-4
 - using for backup 18-17
 - view of last export B-30
- Extension
 - generates recursive calls 22-9

- Extents
 - allocating 8-25
 - allocating for clusters 8-52
 - allocation for index creation 8-34
 - data dictionary views for 8-82
 - displaying free extents 8-84
 - displaying information on 8-84
 - dropped tables and 8-25

F

- Failures
 - during backup 18-13
 - during backups 18-17
 - effects on snapshots 16-10
 - media (disk) 19-27
 - See also Media failure
- “Fast” checkpoint
 - See Checkpoints, “fast”
- Fast refreshes 16-5
- Feedback
 - on ORACLE documentation 1-ix
- Filenames
 - listing 18-6
- Files
 - common parameter file A-6
 - maximum number A-15
 - named in parameter file A-6, A-22
 - OS limit on number open 7-2
 - parameter A-6
 - SQL scripts run at creation A-22
- Files, trace
 - See Trace files
- FIXED_DATE parameter A-20
- Force Checkpoint menu option 5-15
- Force Log Switch menu option 5-15
- Forcing
 - COMMIT or ROLLBACK 15-17
- FOREIGN KEY constraint
 - enabling 8-71
- Format elements
 - D C-9
 - Day C-8
 - G C-9
 - IW C-12
 - IY C-12

Format elements *continued*

- L C-9
- Month C-8
- RM C-7

Free lists

- adding more 24-6
- contention for 24-5
- reducing contention 24-6

Free space

- listing free extents 8-84
- tablespaces and 7-20

FREELISTS storage option

- maximum value A-14

Full backups 18-7

- restoring 19-16

Full database imports A-45

Functions

- recompiling 8-77

G

GC_DB_LOCKS parameter A-20

GC_FILES_TO_LOCKS parameter A-20

GC_LCK_PROCS parameter A-20

GC_ROLLBACK_LOCKS parameter A-21

GC_ROLLBACK_SEGMENTS parameter A-21

GC_SAVE_ROLLBACK_LOCKS

- parameter A-21

GC_SEGMENTS parameter A-21

GC_TABLESPACES parameter A-21

GETMISSES column

- of V\$ROWCACHE table 21-14 to 21-15

GETS column

- of V\$LATCH table 23-11

- of V\$ROWCACHE table 21-14 to 21-15

Global cache control parameters A-5,

- A-20 to A-21

- See also* Instances, multiple

Global database names

- managing 15-3

- SQL*Net communities and 15-3

- viewing 15-6

Global names

- enforcing 15-6

GLOBAL_NAME view B-46

GLOBAL_NAMES parameter A-21

GLOBAL_NAMES parameter *continued*
setting 15-6

GRANT command

- ADMIN option 12-16

- GRANT option 12-17

- object privileges 12-17

- system privileges 12-16

- system privileges and roles 12-16

- when takes effect 12-22

GRANT OPTION 12-17

- revoking 12-19

Grant System Privileges/Roles dialog 12-16

Granting privileges and roles

- GRANT command 12-16

- listing grants 12-26

- shortcuts for object privileges 12-10

GROUP BY clause A-46

Group separator C-9

- default C-6, C-9

Group, redo log

- See* Online redo log, Archived redo log

H

Hardware

- evaluating 1-8

Hash clusters

- altering 8-60

- choosing key 8-57

- controlling space use of 8-57

- creating 8-56

- dropping 8-60

- estimating storage 8-55

- example 8-59

- managing 8-54

- when to use 8-54

- See also* Clusters

HAXLOGHISTORY

- changing 6-4

Hold time

- changing 15-27

HOLD_CURSOR 21-7

HOST

- command in SQL*DBA 5-8

Hot backups

- See* Partial backups

I

I/O

- distributing 2-12, 20-5, 22-4 to 22-6
- reading multiple blocks A-15
- redo log files A-27
- reducing with buffer cache A-12

Icons

- book 1-viii
- in this Guide 1-vii

IDEPTREE view B-46

IDLE column

- of V\$DISPATCHER table 23-4

IFILE parameter A-6

- multiple files A-6
- nested files A-6

IMMEDIATE_GETS column

- of V\$LATCH table 23-12

IMMEDIATE_MISSES column

- of V\$LATCH table 23-12

IMP_FULL_DATABASE role 12-13

Implement

- database design 1-10

Import utility 1-7

- performance A-45
- procedure for using 18-18
- restricted mode and 3-4
- using for recovery 18-17

In-doubt transactions

- committing manually 15-25
- intentionally creating 15-27
- overriding manually 15-17
- resolution of 15-14
- rollback segments and 9-11, 15-17
- rolling back 15-26
- See Transactions, in-doubt

Incomplete recovery

- procedures for 19-22

Incremental exports 18-18

- view of objects exported B-30

IND synonym for USER_INDEXES B-46

Index

- FREELISTS option A-14
 - why estimate size of 8-34
 - why set storage parameters for 8-34
- ## INDEX_HISTOGRAM view B-46

INDEX_STATS view B-47

Indexes 20-4

- altering 8-39
- analyzing statistics 8-62
- auditing 13-6
- creating 8-38
- creating explicitly 8-39
- disabling and dropping constraints and 8-35
- dropped tables and 8-25
- dropping 8-40
- dropping explicitly 8-41
- dropping with constraint 8-41
- estimating size 8-35
- extent allocation for 8-34
- guidelines for managing 8-32
- INITTRANS for 8-33
- limiting per table 8-33
- managing 8-32, 8-41
- MAXTRANS for 8-33
- monitoring space use of 8-40
- PCTFREE for 8-33
- PCTUSED for 8-33
- performance impact A-45
- placement on disk 22-6
- privileges for altering 8-40
- privileges for controlling 12-10
- privileges for creating 8-39
- privileges for dropping 8-41
- snapshots and 16-9
- sort spacemap A-46
- specifying PCTFREE for 8-3
- SQL*Loader and 8-33
- storage parameters 8-9
- tablespace for 8-34
- temporary segments and 8-32
- temporary space and 8-38
- validating structure 8-66
- when to create 8-32
- which tables and columns to index 8-33

Indexes, cluster

- altering 8-52
- creating 8-49
- dropping 8-52
- managing 8-41

INDX column

- of X\$KCB CBH table 21-21

- INDX column *continued*
 - of X\$KCBRBH table 21-18
- INIT.ORA
 - See Parameter files
- INIT_SQL_FILES parameter A-22
 - data dictionary creation and 2-13
- Initial
 - tuning guidelines 2-12
- INITIAL storage parameter 8-6
 - altering 8-24
- Initialization parameters
 - changing A-3
 - controlling archiving with A-24
 - database creation A-17
 - derived A-5
 - detailed descriptions A-8
 - display current settings A-3
 - global cache A-5
 - listing values on startup A-3, A-6
 - LOG_ENTRY_PREBUILD_THRESHOLD parameter 23-13
 - LOG_SIMULTANEOUS_COPIES 23-13
 - rollback segments A-48
 - special parameter file keywords A-6
 - specifying in parameter files A-2
 - tuning for performance A-4
 - upper or lower case A-2
 - variable A-4
 - See also Parameter files
- INITRANS storage parameter
 - altering 8-24
 - default 8-8
 - guidelines for setting 8-8
 - transaction entries and 8-8
- INSERT privilege
 - granting 12-9
 - revoking 12-19
- Installation
 - and creating a database 2-3
 - tuning recommendations for 2-12
- Instance identifier
 - checking 2-9
 - process names and 4-11
- Instance menu
 - Open option 3-9
 - Prevent Connections option 3-9

- INSTANCE_NUMBER parameter A-22
- Instances
 - checkpoint A-27
 - default 2-9
 - failure during online tablespace backup 18-13
 - starting 3-2
 - starting before database creation 2-9
- Integrity constraints
 - disabling 8-70, 8-72
 - disabling on creation 8-71
 - dropping 8-73
 - dropping and disabling 8-35
 - dropping tablespaces and 7-18
 - enabling 8-70
 - enabling on creation 8-71
 - enabling when violations exist 8-71
 - exceptions to 8-73
 - managing 8-70
 - snapshots and 16-4
 - UTLEXCPT.SQL script 8-74
 - violations 8-71
 - when to disable 8-70
- INTERNAL
 - alternatives to 1-5
 - and non-secure connections 1-4
 - connecting as 1-4
 - connecting for shutdown 3-10
 - creating a database as 2-9
 - database recovery and 19-12
 - OSOPER and OSDBA 1-5
 - passwords for 1-4
 - security for 10-6
 - starting an instance as 3-2
- Interrupting
 - media recovery 19-16
- ISO week number C-11
- IW format element C-12
- IY format element C-12

K

- K (kilobytes) 8-6
- Kill
 - user session 4-13
- Kill User Session dialog 4-13

L

- L format element C-9
- LABEL_CACHE_SIZE
 - Trusted ORACLE parameter A-22
- Language
 - default C-5
- Latches
 - See* Redo allocation latch
 - See* Redo copy latches
- Least recently used (LRU) list
 - sequence numbers A-42
- Library cache
 - memory allocation 21-10
 - tuning 21-8
- License agreement
 - complying with 2-8
- LICENSE_MAX_SESSIONS parameter A-23
 - changing while instance runs 11-4
 - setting 11-3
 - setting before database creation 2-8
- LICENSE_MAX_USERS parameter A-23
 - changing while database runs 11-5
 - setting 11-5
 - setting before database creation 2-8
- LICENSE_SESSION_WARNING parameter
 - setting before database creation 2-8
- LICENSE_SESSIONS_WARNING
 - parameter A-24
 - changing while instance runs 11-4
 - setting 11-4
- Licensing
 - complying with license agreement 11-2
 - concurrent usage, using 11-2
 - named user 11-2
 - named user, using 11-4
 - privileges for changing named user
 - limits 11-5
 - privileges for changing session limits 11-4
 - session-based 11-2
 - viewing limits 11-5
- Lightbulb-icon
 - use 1-vii
- Limits
 - concurrent usage 11-2
 - database D-1

- Limits *continued*
 - session, high water mark 11-3
 - See also* Composite limits
 - See also* Resource limits
- List
 - sequence LRU A-42
- List Archive Status menu 5-9
- LIST CHAINED ROWS option 8-66
- LIST keyword
 - displaying parameter values A-3, A-6
- Listener process
 - configuration file 4-4
 - setting MTS_LISTENER_ADDRESS 4-3
- LOADER_COL_INFO view B-48
- LOADER_CONSTRMNT_INFO view B-48
- LOADER_IND_INFO view B-48
- LOADER_INDCOL_INFO view B-48
- LOADER_PARAM_INFO view B-48
- LOADER_TAB_INFO view B-48
- LOADER_TRIGGER_INFO view B-48
- Local currency symbol C-9
- Location
 - rollback segments 9-6
 - transparency 15-11
- Locks
 - in distributed transactions 15-16
 - monitoring 4-9
 - table lock A-42 to A-43
- Log
 - See* Redo log
- Log sequence numbers
 - preserving after recovery 19-26
 - requested during recovery 19-12
 - resetting to 1 19-26
- Log switches
 - checkpoints and 5-13
 - forcing 5-14
 - privileges 5-15
- Log writer process (LGWR)
 - trace file monitoring 4-12
 - tuning 22-4
- LOG_ARCHIVE_BUFFER_SIZE
 - parameter A-24 to A-25
 - setting 17-8 to 17-9
- LOG_ARCHIVE_BUFFERS parameter A-24
 - to A-25

- LOG_ARCHIVE_BUFFERS parameter
 - continued*
 - setting 17-8 to 17-9
- LOG_ARCHIVE_DEST parameter A-25
 - setting 17-12
- LOG_ARCHIVE_FORMAT parameter A-26
 - setting 17-10
- LOG_ARCHIVE_START parameter 17-5, A-26
 - setting 17-5 to 17-6
- LOG_BUFFER parameter A-27
 - setting 23-9
- LOG_CHECKPOINT_INTERVAL
 - parameter A-27
 - guidelines 24-7
 - setting 5-13
- LOG_CHECKPOINT_TIMEOUT
 - parameter A-28
 - guidelines 24-7
 - setting 5-14
- LOG_ENTRY_PREBUILD_THRESHOLD
 - parameter 23-13
- LOG_FILES parameter A-28
 - number of log files and 5-4
- LOG_SIMULTANEOUS_COPIES
 - parameter 23-13, A-29
 - setting 23-10
- LOG_SMALL_ENTRY_MAX_SIZE
 - parameter A-29
 - setting 23-10, 23-13
- LONG datatype 8-12
- LONG RAW datatype 8-12
 - LONG and 8-12
- LRU list
 - sequences A-42

M

- M (megabytes) 8-6
- Maintenance release number
 - See* Version, identifying for ORACLE Server
- Managing
 - auditing 13-1
 - client-server systems 14-1
 - cluster indexes 8-41
 - clustered tables 8-41
 - clusters 8-41

- Managing *continued*
 - database links 15-7
 - indexes 8-32, 8-41
 - object dependencies 8-75
 - ORACLE processes 4-1
 - profiles 11-16
 - roles 12-11
 - rollback segments 9-1
 - sequences 8-29
 - snapshot logs 16-10
 - synonyms 8-31
 - tables 8-15
 - users 11-2
 - views 8-26, 8-29
- Manual archiving A-26
 - in ARCHIVELOG mode 17-6
- “Marked”
 - See* Session, user, “marked” to be killed
- max session memory* statistic 21-16
- MAX_DUMP_FILE_SIZE parameter 4-12, A-29
- MAX_ENABLED_ROLES parameter A-30
 - default roles and 12-14
 - enabling roles and 12-14
- MAX_ROLLBACK_SEGMENTS
 - parameter A-30
- MAXDATAFILES parameter A-15
 - changing 6-4
 - optional use of 7-3
- MAXEXTENTS storage parameter 8-6
 - setting for the data dictionary 8-79
- MAXINSTANCES parameter
 - changing 6-4
- MAXLOGFILES parameter
 - changing 6-4
 - number of log files and 5-3
- MAXLOGMEMBERS parameter
 - changing 6-4
 - number of log files and 5-4
- MAXOPENCURSORS parameter 21-7
- MAXTRANS storage parameter
 - altering 8-24
 - default 8-8
 - guidelines for setting 8-8
 - transaction entries and 8-8
- Media failure (disk failure) 19-27

- Media failure (disk failure) *continued*
 - archived redo log file loss 19-34
 - control file loss 19-34 to 19-35
 - data file loss 19-28
 - online redo log file loss 19-32
 - online redo log group loss 19-30
 - online redo log loss 19-29
- Media recovery
 - applying archived redo logs 19-12
 - cancel-based 19-22
 - change-based 19-22
 - closed database 19-18
 - complete 19-17
 - coordinated time-based distributed database 19-4
 - deciding which action to perform 19-6
 - deciding which files need recovery 19-6
 - determining appropriate operation 19-2
 - distributed databases 19-3
 - errors with redo log files 19-15
 - guidelines and prerequisites 19-2
 - incomplete 19-22
 - interrupting 19-16
 - lost all control files 19-35
 - lost control files 19-34
 - lost mirrored control files 19-35
 - lost of multiple redo log groups 19-33
 - multi-threaded server and 19-12
 - NOARCHIVELOG mode 19-16
 - open database-offline tablespace 19-20
 - performing 19-1, 19-5
 - performing complete 19-18
 - requirements of incomplete 19-26
 - restoring archived redo log files 19-8
 - restoring damaged files 19-8
 - restoring to different devices 19-3
 - resuming 19-16
 - snapshots 19-5
 - snapshots and 19-5
 - starting 19-9
 - startup and 19-12
 - successfully applied redo logs 19-15
 - SYSTEM tablespace 19-20
 - testing 19-2
 - time-based 19-22
 - undamaged tablespaces online 19-10, 19-20

- Media recovery *continued*
 - user errors 19-36 to 19-37
 - using a backup 19-17
 - using a full backup 19-16
 - using SQL*DBA 19-13
 - when completed 19-20, 19-22, 19-25
- Memory
 - real A-45
 - sorting A-45
 - viewing per user 11-25
 - virtual A-4
- Memory allocation
 - for buffer cache 21-18, 21-21
 - for library cache 21-10
 - for shared SQL areas 21-10
 - for users 21-5
 - importance of 21-2
 - for sort areas 24-2
 - tuning 21-2, 21-24
- Migration
 - See Database migration
 - ORACLE Server 1-ii
- MINEXTENTS storage parameter 8-7
 - altering 8-24
- Mirrored control files
 - loss of 19-35
 - why 6-2
- Mirrored redo log files
 - location of 5-2
 - size of 5-3
 - symmetric groups 5-2
- Mirroring
 - control files 2-6
 - online redo log 5-2
 - redo log files 22-4
- MISSES column
 - of V\$LATCH table 23-11
- MLS_LABEL_FORMAT
 - Trusted ORACLE parameter A-30
- MISLABEL datatype 8-13
- Mode
 - multiple-process 4-2
 - single-process 4-2
- Modes
 - archiving A-26
 - exclusive 3-5

- Modes *continued*
 - parallel 3-5
 - restricted 3-4, 3-9
- MONITOR command
 - dynamic performance tables and B-73
 - ROLLBACK option 9-16
 - STAT option A-27
- Monitoring
 - rollback segments 9-16
- Month
 - format element C-8
 - language of names C-8
- Mount Database menu 3-8
- Mounting a database
 - exclusive mode 3-5
 - parallel mode 3-5
- Moving
 - control files 6-4
 - See also* Relocating
- MTS_DISPATCHERS parameter A-30
 - setting initially 4-4
- MTS_LISTENER_ADDRESS parameter A-31
 - setting 4-3
 - starting new dispatchers and 4-8
- MTS_MAX_DISPATCHERS parameter 4-5, A-31
 - setting 4-5
 - tuning dispatchers 23-6
- MTS_MAX_SERVERS parameter A-32
 - setting 4-6
 - tuning servers 23-8
- MTS_SERVERS parameter A-32
 - minimum value 4-6
 - setting 4-6
- MTS_SERVICE parameter A-32
 - DB_NAME and 4-4
 - setting 4-4
- Multi-block reads 22-10
- Multi-threaded server
 - configuring dispatchers 4-4
 - database recovery and 19-12
 - database shutdown and 3-10
 - database startup and 3-2
 - enabling and disabling 4-6 to 4-7
 - operating system authentication
 - restrictions 11-7

- Multi-threaded server *continued*
 - OS role management restrictions 12-25
 - restrictions on OS-role authorization 12-12
 - service name 4-4
 - starting 4-3
 - tuning 23-4
- Multi-threaded server process
 - reducing contention for 23-4
- Multiple-process systems
 - number of processes A-39
 - starting 4-2
- Multiplexed online redo log
 - See* Online redo log, mirroring

N

- Name resolution 8-78
 - location transparency and 15-11
- Named user limits
 - See* Licensing, named user
 - setting initially 2-8
- NAMESPACE column
 - of V\$LIBRARYCACHE table 21-9
- Naming
 - databases 15-3
 - snapshot logs 16-12
 - snapshots 16-3
- National language support
 - specifying default language A-34
 - See also* Language
 - See also* LANGUAGE parameter
- Network communications
 - administration of 15-2
- Network domain
 - control over 15-4
- Network protocol
 - dispatcher for each 4-4
- NEXT storage parameter 8-6
 - setting for the data dictionary 8-79
- NLS_CURRENCY parameter A-33
- NLS_DATE_FORMAT parameter A-33
- NLS_DATE_LANGUAGE parameter A-33, C-8
- NLS_ISO_CURRENCY parameter A-34
- NLS_LANG environment variable C-3
- NLS_LANG parameter A-34

- NLS_LANGUAGE parameter C-5
- NLS_NUMERIC_CHARACTERS
 - parameter A-35, C-9
- NLS_SORT parameter A-36, C-11
- NLS_TERRITORY parameter A-36, C-6
- NOARCHIVELOG mode
 - creating a database A-26
 - data file loss 19-28
 - distributed database backups 18-4
 - partial backups and 18-9
 - setting at database creation 17-3
 - strategies for backups in 18-5
 - taking data files offline in 7-12
 - See also* Archiving
- NOAUDIT command 13-12
 - privileges 13-12
 - schema objects 13-13
 - statements 13-12
- NOCACHE option A-42
- NORESET LOGS option
 - backing up control file 18-16
- NOSORT option 24-4
 - choosing when to use 24-4
 - examples 24-4
 - of CREATE INDEX command 24-4
 - performance benefits 24-4
- NOT NULL constraint
 - See also* Data integrity; Integrity constraint
- NUMBER datatype 8-12
 - range of values stored in 8-12

O

- OBJ synonym for USER_OBJECTS B-48
- Object privileges
 - See also* Privileges, object
- Objects
 - See* Schema objects
- Objects, data dictionary
 - See* Data dictionary
 - DBA B-5
- Offline backups
 - See* Backups
- Offline rollback segments 9-10
 - bringing online 9-10
 - when to use 9-10

- Offline tablespaces 7-9
 - priorities 7-10
 - rollback segments and 9-10
- Online backups
 - See* Backups
- Online redo log
 - active group 19-30
 - applying during recovery 19-14
 - archived group 19-30
 - archiving A-26
 - archiving mode A-25 to A-26
 - block A-27
 - block size A-24
 - creating groups 5-4
 - creating members 5-6
 - current group 19-30
 - dropping groups 5-9
 - dropping members 5-11
 - file size A-27
 - forcing a log switch 5-14
 - guidelines for configuring 5-2
 - inactive group 19-30
 - increasing the size of 24-7
 - location of 5-2
 - loss of 19-29, 19-32
 - loss of all members 19-30
 - loss of group 19-30
 - loss of mirrored members 19-29
 - managing 5-1
 - mirroring 5-2
 - moving files 5-8
 - number of files 5-3, A-26, A-28
 - preserving or resetting log sequence
 - number 19-26
 - privileges for dropping groups 5-10
 - privileges for forcing a log switch 5-15
 - privileges required to add groups 5-5
 - privileges to drop members 5-12
 - renaming files 5-8
 - renaming members 5-7
 - setting checkpoint interval A-27
 - size of 5-2
 - STALE members 5-11
 - status of members 19-30
 - storing separately from data files 7-4
 - unavailable when database is opened 3-7

- Online redo log *continued*
 - viewing filenames 18-6
 - viewing information about 5-16
- Online rollback segments 9-10
 - bringing rollback segments online 9-10
 - taking offline 9-12
 - when new 9-8
- online tablespace backups
 - instance failure during 18-13
- Online tablespaces 7-9
- OPEN_CURSORS parameter A-37
 - allocating more private SQL areas 21-6
 - increasing cursors per session 21-10
- OPEN_LINKS parameter A-37
 - setting 15-10
- OPEN_MOUNTS
 - Trusted ORACLE parameter A-37
- Opening a database
 - after creation 1-10
 - mounted database 3-9
- Operating system
 - auditing with 13-2
 - database administrator's requirements for 1-3
 - deleting data files 7-18
 - limit of number of open files 7-3
 - O/S dependent parameters A-5
 - ORACLE process names 4-10
 - redo log blocks A-24, A-27
 - renaming and relocating files 7-13
 - roles and 12-23, 12-25
 - Security in 10-3
 - tuning 21-4
 - See also* Accounts, operating system
- OPSS\$ account
 - See* OS_AUTHENT_PREFIX parameter
- OPTIMAL storage parameter 9-5
- Optimization
 - trading off cursor space and time A-12
- OPTIMIZER_MODE parameter A-38
- options
 - of the ORACLE Server 1-i
- ORACLE blocks
 - See* Data blocks
- ORACLE Call Interfaces (OCIs)
 - control of parsing and private SQL areas 21-7

- ORACLE Parallel Server
 - See* Parallel Server
- ORACLE Precompilers
 - control of parsing and private SQL areas 21-7
- ORACLE processes
 - Checkpoint (CKPT) 4-13
 - identifying and managing 4-9
 - managing 4-1
 - monitoring 4-9
 - operating system names 4-10
 - trace files for 4-11
- ORACLE RDBMS
 - See* ORACLE Server
- ORACLE Server
 - complying with license agreement 11-2
 - connecting between versions 14-3
 - installing 1-8
 - options 1-i
 - releases 1-11
- Order
 - creating snapshots and snapshot logs 16-10
- OS
 - See* Operating system
- OS_AUTHENT_PREFIX parameter A-38
 - operating system authentication and 11-6
 - setting 11-6
- OS_ROLES parameter A-39
 - operating system authorization and 12-12
 - REMOTE_OS_ROLES and 12-25
 - using 12-23
- OSDBA
 - privileges included in 1-5
- OSOPER
 - privileges included in 1-5

P

- Packages
 - privileges for recompiling 8-77
 - recompiling 8-77
- Paging
 - library cache 21-10
 - reducing 21-4
 - SGA 21-24
- Parallel mode
 - of the database 3-5

Parallel Server

- ALTER CLUSTER..ALLOCATE EXTENT 8-52
- archive log file name format 17-11
- data file upper bound for instances 7-3
- forcing a checkpoint for the local instance 5-15
- licensed session limit and 2-8
- limits on named users and 11-5
- LOG_CHECKPOINT_TIMEOUT and 5-14
- managing rollback segments 9-1
- named users and 2-9
- own rollback segments 9-2
- sequence numbers and 8-30
- session and waning limits 11-3
- specifying thread for archiving 17-8
- switching archiving modes 17-4
- V\$THREAD view 5-16

Parallel Server option 1-i

Parameter

- See also* individual parameter names
- parameter files 3-13

PARAMETER column

- of V\$ROWCACHE table 21-14

Parameter files

- character set of 3-14
- choosing for startup 3-3
- comment character (#) A-3, A-6
- common file A-6
- creating for database creation 2-4
- default for instance startup 3-3
- editing 3-14
- editing before database creation 2-5
- embedded files A-3
- including a SQL file A-22
- line continuation character (\) A-3, A-6
- location of 3-14
- names of A-1
- number of 3-14
- sample of 3-13
- syntax A-3
- using 3-13

- See also* Initialization parameters

Parameters, INIT.ORA

- See* Initialization parameters

PARSED_PIECES view B-48

PARSED_SIZE view B-48

Parsing

- and private SQL areas 21-6
- with the ORACLE Call Interfaces (OCIs) 21-7
- with the ORACLE Precompilers 21-7
- reducing 20-5
- with SQL*Forms 21-7

Partial backups 18-8

Passwords

- changing 11-12, 11-14
- changing for roles 12-14
- for users 10-4
- privileges for changing for roles 12-14
- roles 12-11
- See also* Authentication

Patch release number

- See* Version, identifying for ORACLE Server

PCTTREE storage parameter

- altering 8-24
- block overhead and 8-4
- clustered tables 8-3
- default 8-2
- guidelines for setting 8-2
- indexes 8-3
- non-clustered tables 8-3
- PCTUSED and 8-4

PCTINCREASE storage parameter 8-7

- setting for the data dictionary 8-79

PCTUSED storage parameter

- altering 8-24
- block overhead and 8-4
- default 8-3
- guidelines for setting 8-3
- PCTFREE and 8-4

Performance

- archiving A-24
- dynamic performance tables B-72
- effect of indexes A-46
- effect of SGA size A-12 to A-14
- full database import A-45
- location of data files and 7-4
- monitoring through V\$ tables B-72
- number of database buffers A-12 to A-14
- sorting A-45 to A-46
- tuning archiving 17-8, A-25
- tuning initialization parameters A-4
- tuning process 20-1

- Performance tables
 - See Dynamic performance tables
- PHYRDS column
 - of V\$FILESTAT table 22-3
- physical reads* statistic 21-17
 - calculating hit ratio 21-19, 21-22
- PHYWRDS column
 - of V\$FILESTAT table 22-3
 - of V\$FILESTAT view 22-3
- PIDIAN.SQL 2-14
- PINS column
 - of V\$LIBRARYCACHE table 21-9
- PIPIDL.SQL 2-14
- PISTUB.SQL 2-14
- PL/SQL areas
 - tuning 21-6
- PL/SQL program units
 - dropped tables and 8-25
 - replaced views and 8-29
- Plan
 - the database 1-9
- PLANTABLE execution plan table B-49
- Planning
 - database creation 2-2
- Policies
 - security 10-1
- “Pre-building” redo entries 23-13
- Pre-defined roles
 - See Roles, pre-defined
- Precision
 - NUMBER and FLOAT B-15, B-41, B-69
- Preface 1-i
- Prerequisites
 - for creating a database 2-3
- PRIMARY KEY constraint
 - disabling 8-72
 - dropping associated indexes 8-41
 - enabling 8-71
 - enabling on creation 8-38
 - foreign key references when dropped 8-73
 - indexes associated with 8-38
 - storage of associated indexes 8-38
- Private
 - database links 15-8
 - rollback segments 9-8
 - rollback segments, taking offline 9-13
- Private *continued*
 - synonyms 8-31
- Private rollback segment
 - specifying A-41
- Private SQL areas
 - reuse by multiple SQL statements 21-6
- Privileges 12-2
 - adding data files to a tablespace 7-8
 - adding redo log groups 5-5
 - altering default storage parameters 7-9
 - altering indexes 8-40
 - altering rollback segments 9-10
 - altering sequences 8-30
 - altering snapshot logs 16-13
 - altering snapshots 16-5
 - altering tables 8-25
 - analyzing objects 8-67
 - application developers and 10-8
 - audit object 13-12
 - auditing system 13-11
 - auditing use of 13-8
 - backing up control files 18-17
 - backing up offline tablespaces 18-14
 - backing up online tablespaces 18-12
 - bringing data files offline and online 7-13
 - bringing tablespaces online 7-10
 - cascading revokes 12-20
 - changing named user limit 11-5
 - changing passwords 11-13
 - changing role authorization 12-14
 - changing users 11-14
 - cluster creation 8-51
 - committing in-doubt transactions 15-26
 - CREATE SCHEMA command 8-61
 - creating database links 15-9
 - creating indexes 8-39
 - creating roles 12-13
 - creating rollback segments 9-8
 - creating sequences 8-30
 - creating snapshot logs 16-12
 - creating snapshots 16-3
 - creating synonyms 8-31
 - creating tables 8-23
 - creating tablespaces 7-7
 - creating users 11-11
 - creating views 8-27

Privileges *continued*

- database administrator 1-3
- deleting rows from snapshot logs 16-15
- disabling automatic archiving 17-6
- distributed databases and 15-3
- dropping clusters 8-54
- dropping database links 15-10
- chopping indexes 8-41
- dropping online redo log members 5-12
- dropping profiles 11-21
- dropping redo log groups 5-10
- dropping roles 12-16
- dropping rollback segments 9-16
- dropping sequences 8-31
- dropping snapshot logs 16-16
- dropping snapshots 16-10
- dropping synonyms 8-32
- dropping tables 8-25
- dropping views 8-29
- enabling and disabling resource limits 11-22
- enabling and disabling triggers 8-70
- enabling automatic archiving 17-5
- for changing session limits 11-4
- forcing a checkpoint 5-15
- forcing a log switch 5-15
- granting 12-16 to 12-17
- granting object privileges 12-18
- granting system 12-16
- grouping with roles 12-11
- listing grants 12-27
- managing with procedures 15-13
- managing with synonyms 15-13
- managing with views 15-12
- managing archiving 17-8
- object 12-9
- on selected columns 12-19
- policies for managing 10-4
- privileges to grant 12-18
- privileges to grant roles 12-17
- recompiling packages 8-77
- recompiling procedures 8-77
- recompiling triggers 8-78
- recompiling views 8-77
- refreshing snapshots 16-8
- remote usernames 15-9
- renaming data files of a tablespace 7-14

Privileges *continued*

- renaming data files of several tablespaces 7-15
- renaming objects 8-62
- renaming redo log members 5-9
- replacing views 8-29
- RESTRICTED SESSION 3-4
- revoking 12-18 to 12-19
- revoking ADMIN OPTION 12-19
- revoking GRANT OPTION 12-19
- revoking object 12-20
- revoking object privileges 12-20
- revoking system 12-18
- rolling back in-doubt transactions 15-26
- setting resource costs 11-20
- SQL statements permitted by 12-10
- system 12-2
- taking rollback segments online and offline 9-13
- taking tablespaces offline 7-11
- truncating 8-68
- viewing archive status 17-10
- when revoking object privileges 12-21
- See also* individual privilege names
- See also* Revoking privileges
- Privileges, operating system
 - required for database administrator 1-4
- Procedural database option 1-i
- Procedures
 - auditing 13-6
 - recompiling 8-77
- Process monitor 4-9
- Process priority 20-5, 23-14
- PROCESSES parameter A-39
 - derived parameter values A-48
 - increasing for CKPT 24-7, A-9
 - setting before database creation 2-7
- Processes, ORACLE
 - See* ORACLE processes; Background processes
- Processes, user
 - number of A-39
 - trace files A-49
- Profiles 11-16
 - altering 11-17
 - assigning to users 11-17
 - composite limit 11-18
 - creating 11-16

Profiles *continued*

- default 11-17
- disabling resource limits 11-21
- dropping 11-20
- enabling resource limits 11-21
- listing 11-22
- managing 11-16
- privileges for dropping 11-21
- privileges to alter 11-18
- privileges to set resource costs 11-20
- PUBLIC_DEFAULT 11-17
- setting a limit to null 11-18
- viewing 11-25
- Program global area (PGA)
 - effect of MAX_ENABLED_ROLES on 12-14
- Public
 - database links 15-8
 - synonyms 8-31
- PUBLIC option
 - thread of redo A-47
- Public rollback segments
 - making available for use 9-10
 - specifying A-41
 - taking offline 9-13
 - using by default A-41
- PUBLIC user group
 - dynamic performance table access B-73
 - granting and revoking privileges to 12-22
 - procedures and 12-22
- PUBLIC_DEFAULT profile 11-17
 - dropping profiles and 11-21
- PUBLIC_DEPENDENCY view B-49
- Purging
 - snapshot logs 16-14

Q

Queries

- table-level read locks A-43

Quotas

- listing 11-22
- revoking from users 11-10
- setting to zero 11-10
- temporary segments and 11-10
- unlimited 11-11

Quotas *continued*

- viewing 11-24
- See also* Tablespace quotas

R

RAW datatype 8-12

- VARCHAR2 and 8-12

Read consistency

- export utility and 18-17

Real memory

- sorting A-45

Recompiling

- automatically 8-76
- functions 8-77
- packages 8-77
- procedures 8-77
- triggers 8-77
- views 8-77

Recover Closed Database dialog box 19-9

RECOVER command

- DATABASE parameter 19-10
- DATAFILE parameter 19-12
- TABLESPACE parameter 19-10

Recover In-Doubt Transaction dialog 15-25

Recover Offline Data Files dialog box 19-11

Recover Offline Tablespaces dialog box 19-10

Recoverer process (RECO)

- closing connections established by 15-9
- disabling 15-28
- DISTRIBUTED_TRANSACTIONS parameter and 15-14
- enabling 15-28

Recovery

- planning for distributed databases 15-3
- creating new control files 6-4
- effects of archiving on 17-2
- effects of checkpoints 24-6
- from failure during backup 18-17
- startup with automatic 3-5
- testing distributed transactions 15-27
- user errors 19-37
- using Import for 18-18
- See* Media recovery

Recursive calls 22-9

- and dynamic extension 22-9

- recursive calls* statistic 22-10
- Redo allocation latch 23-10
 - “copying on” 23-10
 - contention for 20-5, 23-13
- Redo copy latches 23-10
 - choosing how many 23-10, 23-13
 - contention for 20-5, 23-13
 - creating more 23-13
- Redo entries
 - “pre-building” 23-13
- Redo log
 - See* Archived redo log
 - See* Online redo log
- Redo log buffers
 - size A-27
 - size of A-27
- Redo log files
 - See* Archived redo log files
 - mirroring 22-4
 - placement on disk 22-4
- redo log space requests* statistic 23-9
- REFERENCES privilege
 - revoking 12-19 to 12-20
- REFERENCES privileges
 - CASCADE CONSTRAINTS option 12-20
- Referential integrity
 - snapshots and 16-8
- Refresh intervals
 - changing 16-8
 - examples 16-6
- Refresh utility 16-2
- Refreshing
 - automatically 16-5
 - manually 16-7
 - snapshots 16-5
- Related publications 1-viii
- Relational design
 - planning 1-9
- Release number
 - See* Releases, identifying for ORACLE Server
- RELEASE_CURSOR 21-7
- Releases
 - how to check 1-12
 - identifying for ORACLE Server 1-11
- RELOADS column
 - of V\$LIBRARYCACHE table 21-9

- Relocating
 - control files 6-4
 - data files 7-13
- Remarks
 - in parameter files A-3, A-6
- REMOTE_OS_AUTHENT parameter A-39
 - setting 11-7
- REMOTE_OS_ROLES parameter A-40
 - setting 12-12, 12-25
- RENAME command 8-61
- Rename Data File dialog 7-13
- Rename Online Redo Log Member dialog box 5-7
- Renaming
 - control files 6-4
 - data files 7-13
 - database 15-5
 - online redo log members 5-7
 - schema objects 8-61
- Replacing
 - views 8-28
- Replicas 16-16
- RESOURCE
 - shortcut for auditing 13-7
- Resource limits
 - altering in profiles 11-17
 - assigning with profiles 11-17
 - composite limits and 11-18
 - costs and 11-19
 - creating profiles and 11-16
 - disabling 11-21
 - distributed databases and 15-3, 15-12
 - enabling 11-21
 - privileges to enable and disable 11-22
 - privileges to set costs 11-20
 - PUBLIC_DEFAULT profile and 11-17
 - service units 11-18
 - setting to null 11-18
 - See also* Profiles
- RESOURCE role 12-13
- RESOURCE_COST view B-49
- RESOURCE_LIMIT parameter A-40
 - enabling and disabling limits 11-21
- Resources
 - See* Profiles

- Responsibilities
 - of a database administrator 1-2
 - of application developers 1-3
 - of database users 1-3
 - of other users 1-3
- Restoring
 - full backup 19-17
- RESTRICTED SESSION privilege
 - restricted mode and 3-4, 3-9
 - session limits and 11-2
- Restricting access to database
 - See* Starting an instance
- Restrictions
 - number of files A-15
- Resuming recovery 19-16
- REVOKE command 12-18
 - when takes effect 12-22
- Revoke System Privileges/Roles dialog 12-18
- Revoking privileges and roles
 - on selected columns 12-19
 - REVOKE command 12-18
 - shortcuts for object privileges 12-10
 - when using operating system roles 12-24
- RM format element C-7
- ROLE_ROLE_PRIVS view B-50
- ROLE_SYS_PRIVS view B-50
- ROLE_TAB_PRIVS view B-50
- Roles
 - ADMIN OPTION and 12-16
 - GRANT OPTION and 12-18
 - application developers and 10-9
 - auditing 13-6
 - authorization 12-11
 - backward compatibility 12-13
 - changing authorization for 12-13
 - changing passwords 12-14
 - CONNECT role 12-13
 - creating 12-11
 - database authorization 12-11
 - DBA 1-5
 - DBA role 12-13
 - default 11-13
 - dropping 12-15
 - EXP_FULL_DATABASE 12-13
 - granting 12-16
 - grouping with roles 12-11

- Roles *continued*
 - IMP_FULL_DATABASE 12-13
 - listing 12-29
 - listing grants 12-27
 - listing privileges and roles in 12-29
 - management using the operating system 12-23
 - managing 12-11
 - multi-byte characters in names 12-11
 - multi-byte characters in passwords 12-11
 - multi-threaded server and 12-12
 - operating system A-39
 - operating system authorization 12-12
 - operating system granting of 12-23, 12-25
 - OS management and the Multi-threaded server 12-25
 - passwords for enabling 12-11
 - pre-defined 1-5, 12-13
 - privileges for changing authorization method 12-14
 - privileges for changing passwords 12-14
 - privileges for creating 12-13
 - privileges to drop 12-16
 - privileges to grant 12-17
 - RESOURCE role 12-13
 - revoking 12-18
 - revoking ADMIN OPTION 12-19
 - security and 10-4
 - unique names for 12-11
 - without authorization 12-13
 - See also* GRANT command;
 - REVOKE command
 - See also* SET ROLE command
- ROLLBACK command
 - FORCE option 15-26
 - forcing 15-17
- Rollback segment
 - acquiring A-41
 - specifying A-41
 - SYSTEM A-41
- Rollback segments
 - acquiring A-48
 - acquiring automatically 9-3, 9-12
 - acquiring on startup 2-7
 - allocating 2-12
 - altering public 9-10

Rollback segments *continued*

- altering storage parameters 9-9
- assigning to transactions 22-11
- auditing 13-6
- AVAILABLE 9-11
- bringing online 9-10
- bringing online automatically 9-12
- bringing online when new 9-8
- checking if offline 9-13
- choosing how many 2-12, 23-3
- choosing size for 2-12
- choosing sizes for 23-3
- contention for 23-2
- creating 9-7, 23-3
- creating after database creation 9-2
- creating public and private 9-3
- deferred 9-17 to 9-18
- displaying deferred 9-17 to 9-18
- displaying information on 9-16
- displaying names of all 9-17
- displaying PENDING OFFLINE segments 9-17
- dropping 9-15
- equally sized extents 9-5
- explicitly assigning transactions to 9-14
- guidelines for managing 9-2
- in-doubt distributed transactions and 15-17
- initial 9-2
- initialization parameters A-48
- invalid status 9-16
- listing extents in 8-84
- location of 9-6
- making available for use 9-10
- managing 9-1
- MINEXTENTS and 8-7
- monitoring 9-5, 9-16
- offline 9-10 to 9-11
- offline status 9-11
- online 9-10
- online status 9-11
- PARTLY AVAILABLE 9-11
- PENDING OFFLINE 9-13
- privileges for dropping 9-16
- privileges to alter 9-10
- privileges to create 9-8
- privileges to take online and offline 9-13

Rollback segments *continued*

- reducing contention for 23-3
- size of 9-3
- status for dropping 9-15
- status or state 9-10
- storage parameters 9-8
- storage parameters and 9-8
- SYSTEM rollback segment 9-2
- taking offline 9-12
- taking tablespaces offline and 7-11
- transactions and 9-14
- use multiple 9-2
- See also* Offline rollback segments
- See also* Online rollback segments
- ROLLBACK_SEGMENTS parameter A-41
 - adding rollback segments to 9-8
 - setting before database creation 2-7
 - unique values A-41
- Roman numerals
 - format mask for C-7
- Row cache A-41
 - See* V\$ROWCACHE table
- Row locking
 - ROW_LOCKING parameter A-42
- ROW_CACHE_CURSORS parameter A-41
- ROW_LOCKING parameter A-42
- ROWID datatype 8-12
- ROWID pseudo-column 8-12
 - changes in 8-13
- Rows
 - chaining across blocks 8-3, 8-66
 - estimating size 8-20 to 8-21
 - ROWID changes 8-13
 - violating integrity constraints 8-71

S

Savepoints

- in-doubt transactions and 15-26

Schema objects

- auditing 13-9
- choosing global names 15-3
- creating global names 15-4
- creating multiple objects 8-60
- default audit options 13-12
- default tablespace for 11-9

Schema objects *continued*

- dependencies between 8-75
 - disabling audit options 13-13
 - enabling audit options on 13-11
 - enforcement of global names 15-6
 - exported B-30
 - general management topics 8-60
 - granting privileges 12-17
 - guidelines for managing 8-2
 - listing by type 8-82
 - listing information 8-81
 - managing 8-1
 - name resolution 8-78
 - owned by dropped users 11-14
 - privileges to access 12-9
 - privileges to rename 8-62
 - remote name resolution 15-11
 - renaming 8-61
 - renaming global names 15-5
 - in a revoked tablespace 11-10
 - when revoking object privileges 12-21
 - revoking privileges 12-19
- security
- and the database administrator 1-3
 - application developers and 10-8
 - auditing policies 10-10
 - authentication of users 10-2
 - connecting as INTERNAL 10-6
 - data 10-3
 - database users and 10-2
 - distributed databases and 15-3
 - establishing policies 10-1
 - for remote objects 15-12
 - general users 10-4
 - location transparency and 15-12
 - multi-byte characters in role names 12-11
 - multi-byte characters in role passwords 12-11
 - multi-byte characters in user names 11-8
 - multi-byte characters in user passwords 11-9
 - operating system security and the database 10-3
 - policies for database administrators 10-6
 - policies for system 10-2
 - policy for users 10-4
 - privilege management policies 10-4
 - protecting the audit trail 13-17

Security *continued*

- public database links and 15-12
 - remote usernames and 15-12
 - REMOTE_OS_ROLES parameter 12-25
 - security administrator 10-2
 - using roles to enforce 10-4
 - See also* Access, database
 - See also* Database security
 - See also* Privileges
- Security officer
- See* Database administrator
- Segments 22-9
- data dictionary 8-79
 - displaying information on 8-83
 - monitoring 9-16
 - name A-41
- Segments, data and index
- default storage parameters 8-9
 - reducing fragmentation 8-7
- Segments, rollback
- See* Rollback segments
- Segments, temporary
- storage parameters 8-10
- Sensitivity
- See* Security
- SEQ synonym for USER_SEQUENCES B-50
- SEQUENCE_CACHE_ENTRIES parameter
- 8-30, A-42
- SEQUENCE_CACHE_HASH_BUCKETS parameter A-43
- Sequences
- altering 8-30
 - auditing 13-6
 - creating 8-29
 - data dictionary cache A-42
 - dropping 8-31
 - initialization parameters 8-30
 - managing 8-29
 - not cached A-42
 - Parallel Server and 8-30
 - privileges for altering 8-30
 - privileges for creating 8-30
 - privileges for dropping 8-31
- SERIALIZABLE parameter A-43
- Server units
- composite limits and 11-18

- Service name
 - for dispatcher in multi-threaded server 4-4
- Session limits, license
 - setting initially 2-8
- session memory* statistic 21-16
- Session monitor 4-9
- Session, user
 - active 4-14
 - inactive 4-14
 - killing 4-13
 - “marked” to be killed 4-15
 - viewing killed sessions 4-15
- SESSION_PRIVS view B-50
- SESSION_ROLES view B-50
- Sessions
 - auditing 13-6
 - auditing connections and disconnections 13-7
 - limits per instance 11-2
 - listing privilege domain of 12-28
 - setting advice for transactions 15-18
 - setting maximum for instance 11-3
 - setting warning limit for instance 11-4
 - viewing current number and high
 - water mark 11-5
 - viewing memory use 11-25
- SESSIONS parameter A-43
- SET ROLE command
 - how password is set 12-11
 - when using operating system roles 12-25
- Set Rollback Segment Offline dialog 9-12
- Set Rollback Segment Online dialog
 - 9-10 to 9-11
- Set Rollback Segment Storage dialog 9-8
- Set Tablespace Offline dialog 7-11, 18-14
- Set Tablespace Online dialog 7-9, 18-14
- Set Tablespace Storage dialog box 7-8
- SET TRANSACTION command
 - examples 22-11
 - USE ROLLBACK SEGMENT option 9-14
- SGA
 - See System Global Area (SGA)
- Shared mode
 - rollback segments and 9-3
- Shared pool
 - ANALYZE command and 8-65
- Shared server processes (Snnn)
 - changing the minimum number of 4-6
 - maximum number of 4-6
 - number to start initially 4-6
 - privileges to change number of 4-7
 - trace files for 4-11
- Shared SQL areas
 - ANALYZE command and 8-65
 - memory allocation 21-10
- SHARED_POOL_SIZE
 - allocating library cache 21-10
- SHARED_POOL_SIZE parameter A-44
 - reducing dictionary cache misses 21-15
 - tuning the shared pool 21-15
- Shortcuts
 - CONNECT, for auditing 13-7
 - object auditing 13-10
 - object privileges 12-10
 - statement level auditing options 13-7
- SHOW command
 - PARAMETERS option A-3
- SHOW SGA command 21-5
- Shut Down menu 3-10
 - Abort Instance option 3-12
 - example of 3-12
 - Immediate option 3-11
 - Normal option 3-10
- Shutdown
 - See Shutting down a database
- SHUTDOWN command 3-10
 - ABORT option 3-12
 - example of 3-13
 - IMMEDIATE option 3-11
 - NORMAL option 3-10
- Shutting down a database 3-1
 - See Shutting down an instance
- Shutting down an instance
 - aborting the instance 3-11
 - connecting and 3-10
 - connecting as INTERNAL 3-10
 - examples of 3-12
 - general procedures 3-10
 - immediately 3-11
 - multi-threaded server 3-10
 - normally 3-10

- SID
 - checking 4-4
- Single-process systems
 - enabling distributed recovery 15-29
 - starting 4-2
- Single-task mode
 - starting 4-2
- SINGLE_PROCESS parameter A-44
 - FALSE in parallel mode A-44
 - setting 4-2
- Size
 - clusters 8-43
 - data file 7-3
 - hash clusters 8-55
 - indexes 8-35
 - rollback segments 9-3
 - tables 8-17, 8-22
- SLEEPS column
 - of V\$LATCH table 23-11
- SMALL_TABLE_THRESHOLD
 - parameter A-44
- Snapshot logs
 - altering storage parameters 16-13
 - creating 16-11
 - dropping 16-15
 - internal creation 16-12
 - managing 16-10
 - managing space use 16-13
 - naming 16-12
 - order of creating 16-10
 - privileges to alter 16-13
 - privileges to create 16-12
 - privileges to delete rows 16-15
 - privileges to drop 16-16
 - purging 16-14
 - reducing space allocated to 16-15
 - storage parameters 8-9, 16-13
 - See also* Snapshots
- “Snapshot too old”
 - OPTIMAL storage parameter and 9-5
- Snapshots
 - alternatives 16-16
 - clustering 16-3
 - creating 16-2
 - declarative constraints and 16-9
 - dropping 16-10
- Snapshots *continued*
 - indexing 16-9
 - integrity constraints and 16-4
 - internals of creation 16-4
 - managing 16-1
 - master tables and 16-9
 - media failure and 16-10
 - media recovery and 19-5
 - naming 16-3
 - preparing to create 16-2
 - privileges to alter 16-5
 - privileges to create 16-3
 - privileges to drop 16-10
 - privileges to manually refresh 16-8
 - refresh intervals 16-6
 - refresh utility 16-2, 16-7
 - refreshing 16-5
 - refreshing automatically 16-5
 - refreshing manually 16-7
 - storage parameters 8-9
 - storage parameters for 16-4
 - underlying tables 16-4
 - See also* Snapshot logs
- Software versions 1-11
- Sort areas
 - memory allocation 24-2
- SORT_AREA_RETAINED_SIZE parameter
 - A-45
 - tuning sorts 24-3
- SORT_AREA_SIZE parameter A-45
 - index creation and 8-32
 - tuning sorts 24-2
- SORT_SPACEMAP_SIZE parameter A-46
- Sorts
 - avoiding on index creation 24-4
 - memory used for A-45
 - specifying non-default C-11
 - storage use A-46
 - tuning 24-2
- sorts (disk) statistic* 24-2
- SOURCE_SIZE view B-50
- Space
 - adding to the database 7-5
 - reducing snapshot logs 16-15
 - SGA A-15, A-28
 - snapshot logs 16-13

- Space *continued*
 - used by indexes 8-40
- SQL areas
 - tuning 21-6
- SQL scripts B-2
 - CATALOG.SQL A-22
 - for creating the data dictionary 2-13
 - See names of individual scripts
 - optional 2-16
 - required 2-13
 - SQL.BSQ A-22
- SQL scripts for data dictionary
 - named in parameter file A-22
- SQL statements
 - circular remote references 15-12
 - disabling audit options 13-12
 - enabling audit options on 13-11
 - privileges required for 12-10
- SQL trace facility
 - when to enable 4-13
- SQL*DBA 1-7
 - monitors and dynamic performance tables B-73
 - operating system account 1-4
 - SHOW SGA command 21-5
 - Start Automatic Archiving dialog 17-5
 - starting 2-9
 - Statistics A-27
- SQL*Forms
 - control of parsing and private SQL areas 21-7
- SQL*Loader 1-7
 - indexes and 8-33
- SQL*Net
 - administration of 15-2
 - communities 15-3
 - distributed databases 15-2
- SQL.BSQ B-2
- SQL.TRACE parameter A-46
 - trace files and 4-11
- STALE status
 - of redo log members 5-11
- STANDARD.SQL 2-14
- Start Automatic Archiving dialog 17-12
- START option
 - automatic archiving A-25
- Start Up Instance dialog box 3-2
- Startup Instance dialog box *continued*
 - example 3-6
 - Force check box 3-5
 - Mount radio button 3-3
 - Nomount radio button 3-3
 - Open radio button 3-4
 - Recover check box 3-5
 - Recover checkbox 19-12
 - Restrict to DBAs check box 3-4
 - specifying a parameter file 3-3
 - specifying database name 3-3
- starting
 - SQL*DBA 2-9
- Starting a database 3-1
 - general procedures 3-2
 - recovering during 19-12
- Starting an instance
 - at database creation 3-3
 - automatically at system startup 3-5
 - connecting as INTERNAL 3-2
 - database closed and mounted 3-3
 - database name A-17
 - database name conflicts and 2-5, 15-5
 - dispatcher processes and 4-4
 - enabling automatic archiving after 17-5
 - enabling automatic archiving at 17-5
 - examples of 3-6
 - exclusive mode 3-5
 - forcing 3-4
 - general procedures 3-2
 - in single or multiple process mode 4-2
 - mounting and opening the database 3-4
 - multi-threaded server and 3-2
 - normally 3-4
 - parallel mode 3-5
 - parameter files 3-3
 - preparing 3-2
 - problems encountered while 3-4
 - recovery and 3-5
 - remote instance startup 3-6
 - restricted mode 3-4
 - specifying database name 3-2
 - troubleshooting 3-7
 - with dedicated servers only 4-2
 - with multi-threaded servers 4-3
 - without mounting a database 3-3

- Startup
 - See Starting a database
- STARTUP command 3-2
 - database name A-17
 - example of 3-6
 - FORCE option 3-5
 - MOUNT option 3-3
 - NOMOUNT option 3-3
 - OPEN option 3-4
 - RECOVER option 3-5, 19-12
 - RESTRICT option 3-4
 - specifying database name 3-3
 - specifying parameter file 3-3
- Statistics
 - session memory* 21-16
 - cache B-87
 - consistent gets* 23-3, 24-6
 - db block gets* 23-3
 - dbwr checkpoints* A-27
 - effect of cache size A-13 to A-14
 - enabling the collection of 21-18
 - max session memory* 21-16
 - recursive calls* 22-10
 - redo log space requests* 23-9
 - sorts (disk)* 24-2
 - sorts (memory)* 24-2
 - timed A-48
 - undo block* 23-2
 - undo header* 23-2
 - updating 8-62
- Steps
 - creating a database 2-4
- STMT_AUDIT_OPTION_MAP table
 - auditing option type codes B-50
- Stop Auto Archive menu option 17-6
- Storage
 - altering tablespaces 7-8
 - quotas and 11-10
 - revoking tablespaces and 11-10
 - unlimited quotas 11-11
 - See also Disk drives
- STORAGE clause
 - examples 22-5
 - of CREATE TABLE command 22-5
- Storage parameters
 - applicable objects 8-5
- Storage parameters *continued*
 - changing settings 8-10
 - data dictionary 8-79
 - default 8-5
 - for the data dictionary 8-79
 - INITIAL 8-6, 8-24
 - INITRANS 8-8, 8-24
 - MAXEXTENTS 8-6
 - MAXTRANS 8-8, 8-24
 - MINEXTENTS 8-7, 8-24
 - NEXT 8-6
 - OPTIMAL (in rollback segments) 9-5
 - PCTFREE 8-24
 - PCTINCREASE 8-7
 - PCTUSED 8-24
 - precedence of 8-10
 - rollback segments 9-8
 - rollback segments and 9-8
 - snapshot logs 16-13
 - snapshots 16-4
 - SYSTEM rollback segment 9-10
 - temporary segment 8-10
- Stored procedures
 - managing privileges 15-13
 - privileges for recompiling 8-77
 - remote object security and 15-13
 - using privileges granted to PUBLIC 12-22
- Stream
 - tape drive 17-9
- “Striping” tables 22-5
 - examples 22-5
- Swapping
 - library cache 21-10
 - reducing 21-4
 - SGA 21-24
- Switch redo log file
 - See Log switch
- SYN synonym for USER_SYNONYMS B-50
- Synonyms
 - auditing 13-6
 - creating 8-31
 - displaying dependencies of 8-83
 - dropped tables and 8-25
 - dropping 8-31
 - managing 8-31
 - managing privileges 15-13

- Synonyms *continued*
 - private 8-31
 - privileges for creating 8-31
 - privileges for dropping 8-32
 - public 8-31
 - remote object security and 15-13
- SYS
 - dynamic performance tables and B-73
 - initial password 1-6
 - objects owned 1-6
 - policies for protecting 10-6
 - privileges 1-6
 - user 1-6
- SYS.AUD\$
 - See* Audit trail
 - creating and deleting 13-4
- SYSTEM
 - initial password 1-6
 - objects owned 1-6
 - policies for protecting 10-6
 - user 1-6
- System Change Number (SCN)
 - checking for a data file 7-20
 - in-doubt transactions and 15-26
 - use in distributed recovery 19-5
- System Global Area (SGA) 21-5
 - buffer areas A-12, A-27
 - buffer cache A-12
 - sequence cache A-42
 - size of A-12, A-15, A-28, A-48
 - tuning A-13 to A-14
- System privileges
 - See* Privileges, system
- SYSTEM rollback segment 9-2
 - altering storage parameters of 9-10
 - not specified in parameter file A-41
 - remaining online always 9-10
 - See also* Rollback segments
- SYSTEM tablespace
 - backing up 18-13
 - cannot drop 7-17
 - initial rollback segment 9-2
 - non-data dictionary tables and 8-16
 - recovery of 19-20
 - renaming or moving data files in 7-13
 - restrictions on taking offline 7-12

- SYSTEM tablespace *continued*
 - when created 7-5
- SYSTEM_PRIVILEGE_MAP view B-50

T

- Table “striping”
 - examples 22-5
- Table snapshots
 - referential integrity and 16-8
 - See* Snapshots
- TABLE_PRIVILEGE_MAP table
 - access privilege codes B-51
- TABLE_PRIVILEGES view B-51
- Tables
 - allocating extents 8-25
 - altering 8-23 to 8-24
 - analyzing statistics 8-62
 - auditing 13-7
 - clustered tables storage 8-49
 - copying in distributed databases 16-16
 - creating 8-22
 - data dictionary B-1
 - designing before creating 8-15
 - DML locks A-19
 - dropping 8-25
 - estimating initial size 8-17
 - estimating size 8-16, 8-22
 - free lists A-14
 - guidelines for managing 8-15, 8-17
 - increasing column length 8-24
 - limiting extents in 8-17
 - limiting indexes on 8-33
 - location 8-23
 - location of 8-15
 - managing 8-15
 - maximum number of D-4
 - placement on disk 22-6
 - planning for large tables 8-17
 - privileges for creation 8-23
 - privileges for dropping 8-25
 - privileges to alter 8-25
 - schema of clustered 8-50
 - separating from indexes 8-17
 - specifying PCTFREE for 8-3
 - specifying tablespace 8-15, 8-23

Tables *continued*

- storage parameters 8-9
- SYSTEM tablespace and 8-16
- temporary space and 8-17
- truncating 8-67
- underlying snapshot logs 16-12
- underlying snapshots 16-4
- validating structure 8-66

Tables, clustered

- altering 8-52
- creating 8-49
- dropping 8-52
- managing 8-41

Tables, hash clustered

- creating 8-56
- managing 8-54

TABLESPACE clause

- examples 22-5
- of CREATE TABLE command 22-5 to 22-6

Tablespace quotas

- why to assign 7-5

Tablespaces

- adding data files to 7-7
- altering availability 7-9
- altering storage settings 7-8
- assigning defaults for users 11-9
- assigning user quotas 7-5
- auditing 13-7
- backing up offline 18-13
- backing up online 18-9
- backing up several online 18-12
- bringing online and offline 7-9
- checking default storage parameters 7-19
- creating 7-5
- creating additional 7-5
- default quota 11-10
- default storage parameters for 8-9
- default temporary 11-10
- dropping 7-17
- frequency of backups 18-3
- guidelines for managing 7-2
- listing files of 7-19
- listing free space in 7-20
- location 7-4
- managing 7-1
- maximum number of D-4

Tablespaces *continued*

- monitoring 7-19
- privileges for backing up 18-12
- privileges for creating 7-7
- quotas for users 11-10
- revoking from users 11-10
- setting default storage parameters for 7-4
- SYSTEM tablespace 7-5
- taking offline immediately 7-10
- taking offline normal 7-10
- taking offline temporarily 7-10
- temporary 11-10
- unlimited quotas 11-11
- using multiple 7-2
- viewing quotas 11-24

TABS synonym for USER_TABLES B-51

Taking offline

- tablespaces 7-10

Tape archiving destination A-25

Tape drives

- streaming for archiving 17-9

Temporary segments 22-9

- index creation and 8-32

TEMPORARY_TABLE_LOCKS parameter A-47

Territory C-6

Test

- security for databases 10-8

Text

- conventions in this Guide 1-vi

Thread A-47

- disabled A-47
- public A-47

THREAD parameter A-47

Time-based recovery

- coordinated in a distributed database 19-4
- procedure for 19-23

TIMED_STATISTICS parameter A-48

- file read/write statistics B-77

Tip

- executing OS commands within SQL*DBA 5-8
- object privilege shortcut 12-10
- setting for distributed transactions 15-18
- shortcuts for auditing objects 13-10
- statement auditing shortcut 13-7
- using database links 15-7

- TO_CHAR function
 - group separator and C-9
 - spelling of days and months C-8
- TO_DATE function
 - spelling of days and months C-8
- TO_NUMBER function group separator and C-9
- TOTALQ column
 - of V\$QUEUE table 23-5, 23-7
- Trace facility
 - See SQL trace facility
- Trace files
 - ALERT file A-9
 - control file backups to 18-15
 - destination A-9, A-49
 - location 4-12
 - location of A-9
 - log writer 4-12
 - maximum size A-29
 - size 4-12
 - using 4-11 to 4-12
 - when written 4-12
- Transaction entries
 - guidelines for storage 8-8
- Transactions
 - assigning rollback segments to 22-11
 - assigning to specific rollback segment 9-14
 - concurrent A-48
 - data locking A-19, A-43
 - in-doubt 15-15
 - manual overriding in-doubt 15-18
 - older versions in 14-3
 - rollback segments A-48
 - rollback segments and 9-14
 - updates A-43
- TRANSACTIONS parameter A-41, A-48
 - DML locks A-19
 - using 9-2
- TRANSACTIONS_PER_ROLLBACK_SEGMENT parameter A-41, A-48
 - using 9-2
- Transparency
 - location 15-11
 - name resolution and 15-11
- Triggers
 - auditing 13-7, 13-20
- Triggers *continued*
 - disabling 8-69
 - dropped tables and 8-25
 - enabling 8-69
 - examples 13-20
 - privileges for controlling 12-10
 - privileges for enabling and disabling 8-70
 - privileges for recompiling 8-78
 - recompiling 8-77
- Troubleshooting
 - distributed transactions 15-15
- TRUNCATE
 - effects on snapshots 16-9
- TRUNCATE command 8-67
 - DROP STORAGE option 8-68
 - REUSE STORAGE option 8-68
- Truncating
 - clusters 8-67
 - privileges for 8-68
 - tables 8-67
- Trusted ORACLE
 - ALL_LABELS view B-10
 - ALL_MOUNTED view B-10
 - auditing 13-1
 - controlling database access 11-1
 - creating a database using 2-3
 - data dictionary B-1
 - database links 15-9
 - database migration to 2-3
 - database startup and 3-2
 - establishing security policies 10-1
 - LOG_CHECKPOINT_INTERVAL and 5-14
 - managing rollback segments 9-1
 - managing schema objects 8-1
 - managing tablespaces and data files 7-1
 - managing users and resources 11-1
 - role management 12-1
 - shutting down a database 3-10
 - special datatypes 8-13
 - system privileges 12-1
- Tuning
 - applications 20-3
 - archiving 17-8
 - buffer cache 21-17
 - checkpoints 24-6
 - contention 20-3, 23-1

- Tuning *continued*
 - data dictionary cache 21-14
 - I/O 20-3
 - initially 2-12
 - library cache 21-8
 - memory allocation 20-3, 21-2, 21-24
 - method for 20-2
 - operating system 21-4
 - setting goals for 20-2
 - sorts 24-2
 - SQL and PL/SQL areas 21-6
 - SQL statements 20-3
 - top tips for 20-4
- Two-phase commit
 - effects of failure during 15-16
 - failures in 15-15
 - forcing to fail 15-27
 - locked resources and 15-16
 - testing recovery 15-27

U

- undo block* statistic 23-2
- undo header* statistic 23-2
- UNIQUE key constraints
 - disabling 8-72
 - dropping associated indexes 8-41
 - enabling 8-71
 - enabling on creation 8-38
 - foreign key references when dropped 8-73
 - indexes associated with 8-38
 - storage of associated indexes 8-38
- UNLIMITED TABLESPACE privilege
 - overriding other quotas 11-11
- UNLIMITED TABLESPACE privileges
 - assigning 11-11
- UPDATE privilege
 - revoking 12-19
- Updates
 - serializable A-43
 - transaction processing option A-42
- User error
 - recovery from 19-36 to 19-37
- User group
 - PUBLIC 12-22
- User processes, trace files A-49

- User session
 - See* Session, user
- USER_AUDIT_OBJECT view B-52
- USER_AUDIT_SESSION view B-53
- USER_AUDIT_STATEMENT view B-54
- USER_AUDIT_TRAIL view B-55
- USER_CATALOG view B-56
 - CAT synonym B-18
- USER_CLU_COLUMNS view B-57
- USER_CLUSTERS view B-57
 - CLU synonym B-18
- USER_COL_COMMENTS view B-57
- USER_COL_PRIVS view B-58
- USER_COL_PRIVS_MADE view B-58
- USER_COL_PRIVS_RECD view B-58
- USER_CONS_COLUMNS view B-59
- USER_CONSTRAINTS view B-59
- USER_DB_LINKS view B-59
- USER_DEPENDENCIES view B-60
- USER_DUMP_DEST parameter 4-12, A-49
- USER_ERRORS view B-60
- USER_EXTENTS view B-60
- USER_FREE_SPACE view B-60
- USER_IND_COLUMNS view B-62
- USER_INDEXES view B-61
 - filling with data 8-63
 - IND synonym B-46
- USER_OBJ_AUDIT_OPTS view B-63
- USER_OBJECT_SIZE view B-62
- USER_OBJECTS view B-62
 - OBJ synonym B-48
- USER_RESOURCE_LIMITS view B-63
- USER_ROLE_PRIVS view B-64
- USER_SEGMENTS view B-64
- USER_SEQUENCES view B-65
 - SEQ synonym B-50
- USER_SNAPSHOT_LOGS view B-66
- USER_SNAPSHOTS view B-66
- USER_SOURCE view B-67
- USER_SYNONYMS view B-67
 - SYN synonym B-50
- USER_SYS_PRIVS view B-67
- USER_TAB_COLUMNS
 - COLS synonym B-18
- USER_TAB_COLUMNS view B-69
 - filling with data 8-63

- USER_TAB_COMMENTS view B-69
- USER_TAB_PRIVS view B-70
- USER_TAB_PRIVS_MADE view B-70
- USER_TAB_PRIVS_RECD view B-70
- USER_TABLES view B-68
 - filling with data 8-63
- USER_TABLESPACES view B-69
- USER_TRIGGER_COLS view B-71
- USER_TRIGGERS view B-71
- USER_TS_QUOTAS view B-71
- USER_USERS view B-72
- USER_VIEWS view B-72
- Username
 - SYS and SYSTEM 1-6
- Users
 - altering 11-11
 - assigning profiles to 11-17
 - assigning tablespace quotas 7-5
 - assigning unlimited quotas for 11-11
 - authenticating 11-6
 - authenticating with the database 11-7
 - authenticating with the operating system 11-6
 - authentication 10-2, 10-4
 - changing authentication method 11-12
 - changing default roles 11-13
 - changing passwords 11-12
 - composite limits and 11-18
 - creating 11-7
 - default tablespaces 11-9
 - dropped roles and 12-15
 - dropping 11-14
 - dropping profiles and 11-21
 - end-user security policies 10-4
 - enrolling 1-10
 - in a newly created database 2-11
 - listing 11-22
 - listing privileges granted to 12-27
 - listing roles granted to 12-27
 - managing 11-2
 - multi-byte characters in names 11-8
 - multi-byte characters in passwords 11-9
 - multiple A-19
 - objects after dropping 11-14
 - policies for managing privileges 10-4
 - privileges for changing passwords 11-14
 - privileges for creating 11-11

- Users *continued*
 - privileges for dropping 11-15
 - security and 10-2
 - security for general users 10-4
 - specifying user names 11-8
 - tablespace quotas 11-10
 - viewing information on 11-23
 - viewing memory use 11-25
 - viewing tablespace quotas 11-24
 - why use unique user names 11-4
- Users, database
 - multiple A-19
- Utilities
 - Export 1-7
 - for the database administrator 1-7
 - Import 1-7
 - SQL*DBA 1-7
 - SQL*Loader 1-7
- UTLBSTAT.SQL 2-16, B-4
- UTLCHAIN.SQL 2-16, 8-66, 22-7, B-4
- UTLDTREE.SQL 2-16, B-4
- UTLESTAT.SQL 2-16, B-4
- UTLEXCPT.SQL 2-16, B-4
 - running 8-74
- UTLEXP6.SQL 2-16, B-5
- UTLIDXSD.SQL 2-16, B-4
- UTUDXSO.SQL B-4
- UTLIDXSS.SQL B-4
- UTLLOCKT.SQL 2-16, 4-10, B-4
- UTLMAIL.SQL 2-16, B-4
- UTLMONTR.SQL 2-16, B-4, B-73
- UTLOIDX.SQL 2-16
- UTLSAMP.SQL 2-16
- UTLSAMPL.SQL B-4
- UTLSIDX.SQL 2-16
- UTLXPLAN.SQL 2-16, B-4

V

- V\$ tables
 - See also* Dynamic performance tables
- V\$ACCESS view B-73
- V\$ARCHIVE view B-73
 - using 17-9
- V\$BACKUP view B-73
 - using 18-11

V\$BGPROCESS view B-74
 V\$CIRCUIT view B-74
 V\$DATABASE view B-75
 using 17-9
 V\$DATAFILE view B-75
 using 18-6
 using in tuning 22-3
 V\$DB_OBJECT_CACHE view B-76
 V\$DBFILE view B-75
 V\$DISPATCHER view B-77
 controlling dispatcher process load 4-7
 using 23-4
 V\$ENABLEDPRIVS view B-77
 V\$FILESTAT table
 PHYWRFS column 22-3
 V\$FILESTAT view B-77
 PHYRDS column 22-3
 PHYWRFS column 22-3
 using 22-3
 V\$FIXED_TABLE view B-77
 V\$INSTANCE view B-78
 V\$LATCH view B-78
 GETS column 23-11
 IMMEDIATE_GETS column 23-12
 IMMEDIATE_MISSES column 23-12
 MISSES column 23-11
 SLEEPS column 23-11
 using 23-11
 V\$LATCHHOLDER view B-78
 V\$LATCHNAME view B-78
 V\$LIBRARYCACHE view B-79
 NAMESPACE column 21-9
 PINS column 21-9
 RELOADS column 21-9
 using 21-9
 V\$LICENSE view B-80
 using 11-5
 V\$LOADCSTAT view B-80
 V\$LOADTSTAT view B-80
 V\$LOCK view B-81
 V\$LOG view B-83
 using 5-16, 17-9
 V\$LOG_HISTORY view B-83
 V\$LOGFILE view B-83
 using 5-16, 18-6
 V\$LOGHIST view B-84
 V\$NLS_PARAMETERS view B-84
 V\$OPEN_CURSOR view B-84
 V\$PARAMETER view B-84
 V\$PROCESS view B-85
 V\$QUEUE view B-85
 controlling dispatcher process load 4-7
 examining wait times 23-5
 identifying contention 23-7
 V\$RECOVER_FILE view B-86
 V\$RECOVERY_FILE view
 using 19-6
 V\$RECOVERY_LOG view B-85
 V\$REQDIST view B-86
 V\$RESOURCE view B-86
 V\$ROLLNAME view B-86
 finding PENDING OFFLINE segments 9-17
 V\$ROLLSTAT view B-87
 finding PENDING OFFLINE segments 9-17
 V\$ROWCACHE view B-88
 data dictionary cache performance
 Statistics 21-14
 GETMISSES column 21-14 to 21-15
 GETS column 21-14 to 21-15
 PARAMETER column 21-14
 using 21-14
 V\$SECONDARY view B-88
 V\$SESS_IO view B-91
 V\$SESSION view B-89
 using 4-15
 V\$SESSION_WAIT view B-91
 V\$SESSTAT view B-91
 using 21-16
 V\$SGA view B-91
 V\$SGASTAT view B-91
 V\$SHARED_SERVER view B-92
 V\$SQLAREA view B-93
 V\$SQLTEXT view B-94
 V\$STATNAME view B-94
 V\$SYSLABEL view B-94
 V\$SYSSTAT view B-95
 examining recursive calls 22-10
 tuning sorts 24-2
 using 21-17
 viewing redo buffer space 23-9
 V\$THREAD view B-95
 using 5-16

- V\$TIMER view B-95
- V\$TRANSACTION view B-95
- V\$TYPE_SIZE view B-96
- V\$VERSION view B-96
- V\$WAITSTAT view B-96
 - reducing free list contention 24-5
 - viewing rollback segment contention 23-2
- VALIDATE STRUCTURE option 8-66
- VARCHAR2 datatype 8-11
 - space use of 8-11
- Variable parameters A-4
- Version 5
 - communicating with 14-3
- Version 6
 - communicating with 14-3
- Version 7
 - See ORACLE Server
- Versions
 - communicating with older versions 14-3
 - of other Oracle software 1-12
 - last export B-29 to B-30
 - See Releases
- Views
 - auditing 13-7
 - creating 8-26
 - creating with errors 8-27
 - data dictionary B-5
 - DBA B-5
 - displaying dependencies of 8-83
 - dropped tables and 8-25
 - dropping 8-29
 - expanding queries at creation 8-27
 - FOR UPDATE clause and 8-26
 - managing 8-26, 8-29
 - managing privileges with 15-12
 - ORDER BY clause and 8-26
 - privileges 8-27
 - privileges for dropping 8-29
 - privileges for recompiling 8-77
 - privileges to replace 8-29
 - recompiling 8-77
 - remote object security and 15-12
 - replacing 8-28
 - wildcards in 8-27
 - WITH CHECK OPTION 8-26
- Violating integrity constraints 8-71

- Virtual tables
 - X\$KCBCBH table 21-21
 - X\$KCBRBH table 21-18

W

- WAIT column
 - of V\$QUEUE table 23-5, 23-7
- warning
 - archiving mode for first backup 18-5
 - changing data dictionary storage parameters 8-79
 - consistency and Export backups 18-18
 - data filenames missing from control file 18-6
 - data in dropped tablespace is unrecoverable 7-17
 - disabling audit options 13-12
 - enabling auditing 13-10
 - ending online backups 18-11
 - keeping symmetric online redo log groups 5-2
 - setting the CONTROL_FILES parameter 2-6
 - use mirrored control files 6-2
- Warning icon
 - use Preface vii
- Week numbers C-11

X

- X\$KCBCBH table A-14
 - buffer cache performance statistics 21-21
 - COUNT column 21-21
 - enabling use of 21-21
 - INDX column 21-21
- X\$KCBRBH table A-13
 - buffer cache performance statistics 21-18
 - COUNT column 21-18
 - enabling use of 21-18
 - INDX column 21-18