
Enterprise DBA Part 1A: Architecture and Administration

Volume 1 • Student Guide

30049GC10

Production 1.0

August 1999

M09005

ORACLE®

Authors

Bruce Ernst
Hanne Rue Rasmussen
Ulrike Schwinn
Vijay Venkatachalam

Technical Contributors and Reviewers

David Austin
Ben van Balen
Gerry Batista
Doug Bridges
Sandra Cheevers
Ralf Durben
Ari Fyhr
Joel Goodman
Scott Gossett
Lex de Haan
Tony Holbrook
Heike Hundt
Christine Jeal
Dominique Jeunot
Thomas Kerepes
Steven King
Pierre Labrousse
Dean Margolese
Jean-Marie Misztela
Tigger Newman
Howard Ostrow
Hans Proetzi
Gary Purcell
Shankar Raman
Donalyn Selinsky
Roger Simon
James Spiller
Ramonito Te
Sabine Teuber
Jean-Francois Verrier
Norbert Wittje

Publisher

Sherry Polm

Copyright © Oracle Corporation, 1999. All rights reserved.

This documentation 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 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 for commercial computer software and shall be deemed to be Restricted Rights software under Federal law, as set forth in subparagraph (c) (1) (ii) of DFARS 252.227-7013, Rights in Technical Data and Computer Software (October 1988).

This material or any portion of it may not be copied in any form or by any means without the express prior written permission of Oracle Corporation. Any other copying is a violation of copyright law and may result in civil and/or criminal penalties.

If this 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 Alternate III (June 1987).

The information in this document is subject to change without notice. If you find any problems in the documentation, please report them in writing to Education Products, Oracle Corporation, 500 Oracle Parkway, Box SB-6, Redwood Shores, CA 94065. Oracle Corporation does not warrant that this document is error-free.

SQL*Loader, SQL*Net, SQL*Plus, Net8, Oracle Call Interface, Oracle7, Oracle8, Oracle8i, Developer/2000, Developer/2000 Forms, Designer/2000, Oracle Enterprise Manager, Oracle Parallel Server, PL/SQL, Pro*C, Pro*C/C++, and Trusted Oracle are trademarks or registered trademarks of Oracle Corporation.

All other products or company names are used for identification purposes only and may be trademarks of their respective owners.

Contents

Preface

- Profile xvii
- Related Information xix
- Typographic Conventions xx

Introduction

- Objectives I-2
- Oracle8i Enterprise Edition I-3
- Database Administrator Tasks I-5
- Course Schedule I-6

Lesson 1: Oracle Architectural Components

- Objectives 1-2
- Overview 1-3
- Oracle Database Files 1-5
- Other Key Files 1-6
- Oracle Instance 1-7
- Processing a SQL Statement 1-9
- Connecting to a Database 1-10
- Processing a Query 1-12
- The Shared Pool 1-14
- Database Buffer Cache 1-16
- Program Global Area 1-17
- Processing a DML Statement 1-19
- Redo Log Buffer 1-21
- Rollback Segment 1-22
- COMMIT Processing 1-23
- Log Writer 1-25
- Other Instance Processes 1-26
- Database Writer 1-27
- SMON: System Monitor 1-28
- PMON: Process Monitor 1-30
- Archiving 1-31
- Summary 1-33

Lesson 2: Getting Started with the Oracle Server

- Objectives 2-2
- Overview 2-3
- Oracle Universal Installer 2-4
- Validating Privileged Users 2-8
- Oracle Enterprise Manager 2-15
- Summary 2-30

Lesson 3: Managing an Oracle Instance

- Objectives 3-2
- Overview 3-3
- Stages in Startup and Shutdown 3-10
- Starting Up the Instance 3-13
- Changing Database Availability 3-16
- Opening a Database in Read-Only Mode 3-17
- Shutting Down 3-18
- Getting and Setting Parameter Values 3-22
- Managing Sessions 3-29
- Summary 3-36

Lesson 4: Creating a Database

- Objectives 4-2
- Overview 4-3
- Preparing the Operating System 4-4
- Creating a Database 4-10
- Using the Database Configuration Assistant 4-11
- Creating a Database Manually 4-15
- Summary 4-29

Lesson 5: Creating Data Dictionary Views and Standard Packages

- Objectives 5-2
- Overview 5-3
- Data Dictionary Overview 5-4
- Data Dictionary Contents 5-5

- Base Tables and Data Dictionary Views 5-6
- How the Data Dictionary Is Used 5-7
- Data Dictionary View Categories 5-8
- Data Dictionary Examples 5-10
- Dynamic Performance Views 5-11
- Stored Program Units 5-12
- Stored PL/SQL Program Units 5-14
- Packages 5-16
- Executing a PL/SQL Program Unit 5-17
- Package Specification and Body 5-18
- Oracle-Supplied Packages 5-19
- Obtaining Information 5-20
- Troubleshooting 5-22
- Constructing the Data Dictionary 5-23
- Administrative Scripts 5-25
- Built-in Package Example 5-27
- Triggers 5-29
- Parts of a Trigger 5-31
- Trigger Example 5-33
- Summary 5-34

Lesson 6: Maintaining the Control File

- Objectives 6-2
- The Use of the Control File 6-3
- Control File Contents 6-4
- Multiplexing the Control File 6-6
- Guidelines for Control Files 6-7
- Obtaining Information About the Control File 6-9
- Summary 6-11

Lesson 7: Maintaining Redo Log Files

- Objectives 7-2
- Overview 7-3
- Using Online Redo Files 7-4

- LGWR, Log Switches, and Checkpoints 7-6
- Archiving Redo Log Files 7-8
- Obtaining Log and Archive Information 7-11
- Controlling Log Switches and Checkpoints 7-18
- Multiplexing and Maintaining Members and Groups 7-21
- Relocating or Renaming Online Redo Log Files 7-24
- Dropping Online Redo Log Groups and Members 7-26
- Planning Online Redo Logs 7-30
- Troubleshooting 7-32
- Using LogMiner 7-33
- Summary 7-40

Lesson 8: Managing Tablespaces and Data Files

- Objectives 8-2
- Overview 8-3
- Database Storage Hierarchy 8-4
- SYSTEM and Non-SYSTEM Tablespaces 8-7
- Creating Tablespaces 8-8
- Space Management in Tablespaces 8-12
- Locally Managed Tablespaces 8-13
- Temporary Tablespace 8-15
- Changing the Storage Settings 8-18
- Taking Tablespaces Offline or Online 8-20
- Read-Only Tablespaces 8-23
- Dropping Tablespaces 8-26
- Resizing a Tablespace 8-29
- Enabling Automatic Resizing of Data Files 8-30
- Manually Resizing Data Files 8-33
- Adding Data Files to a Tablespace 8-34
- Moving Data Files 8-36
- Data Dictionary Information 8-39
- Guidelines 8-40
- Summary 8-42

Lesson 9: Storage Structure and Relationships

- Objectives 9-2
- Overview 9-3
- Types of Segments 9-4
- Storage Clause Precedence 9-8
- Extent Allocation and Deallocation 9-9
- Used and Free Extents 9-10
- Using Block Space Utilization Parameters 9-11
- Obtaining Information About Storage Structures 9-16
- Querying DBA_SEGMENTS 9-17
- Querying DBA_EXTENTS 9-18
- Querying DBA_FREE_SPACE 9-19
- Planning the Location of Segments 9-20
- Summary 9-22

Lesson 10: Managing Rollback Segments

- Objectives 10-2
- Overview 10-3
- Rollback Segments 10-4
- Using Rollback Segments with Transactions 10-8
- Creating Rollback Segments 10-12
- Maintaining Rollback Segments 10-18
- Obtaining Rollback Segment Information 10-26
- Planning Rollback Segments 10-32
- Troubleshooting Rollback Segment Problems 10-34
- Summary 10-40

Lesson 11: Managing Tables

- Objectives 11-2
- Overview 11-3
- Oracle Data Types 11-7
- Creating a Table 11-17
- Controlling Space Used by Tables 11-25
- Retrieving Table Information 11-41
- Summary 11-47

Lesson 12: Managing Indexes

- Objectives 12-2
- Overview 12-3
- Creating Indexes 12-12
- Reorganizing Indexes 12-22
- Dropping Indexes 12-29
- Obtaining Index Information 12-31
- Summary 12-33

Lesson 13: Maintaining Data Integrity

- Objectives 13-2
- Overview 13-3
- Integrity Constraints 13-5
- Implementing Constraints 13-14
- Maintaining Constraints 13-19
- Getting Constraint Information 13-26
- Summary 13-29

Lesson 14: Loading Data

- Objectives 14-2
- Overview 14-3
- Loading Data Using Direct-Load Insert 14-5
- Loading Data Using SQL*Loader 14-8
- Direct Path Loading 14-28
- Summary 14-29

Lesson 15: Reorganizing Data

- Objectives 15-2
- Overview 15-3
- Transportable Tablespaces 15-23
- Transporting a Tablespace 15-25
- Exporting and Importing Metadata 15-26
- Transporting a Tablespace 15-28
- Transportable Tablespace Uses 15-29

Transportable Tablespaces and Schema Objects 15-30

Checking the Transport Set 15-31

Summary 15-32

Lesson 16: Managing Password Security and Resources

Objectives 16-2

Overview 16-3

Administering Passwords 16-5

Altering and Dropping a Profile 16-20

Controlling Usage of Resources 16-24

Viewing Password and Resource Limits Information 16-31

Summary 16-33

Lesson 17: Managing Users

Objectives 17-2

Overview 17-3

Creating New Database Users 17-6

Altering and Dropping Database Users 17-14

Dropping Users 17-17

Monitoring Information About Users 17-18

Summary 17-20

Lesson 18: Managing Privileges

Objectives 18-2

Overview 18-3

System Privileges 18-4

Granting System Privileges 18-6

Password File Authentication 18-9

Displaying System Privileges 18-11

Revoking System Privileges 18-14

Object Privileges 18-17

Granting Object Privileges 18-18

Displaying Object Privileges 18-20

Revoking Object Privileges 18-21

Auditing Guidelines 18-25
Using Database Auditing 18-29
Viewing Auditing Results 18-36
Summary 18-37

Lesson 19: Managing Roles

Objectives 19-2
Overview 19-3
Creating and Modifying Roles 19-5
Assigning Roles 19-11
Controlling Availability of Roles 19-13
Displaying Role Information 19-22
Using Fine-Grained Access Control 19-23
Summary 19-25

Lesson 20: Using National Language Support

Objectives 20-2
Overview 20-3
Choosing a Database and a National Character Set 20-5
Specifying Language-Dependent Behavior 20-11
NLS Parameters and SQL Functions 20-19
NLS Parameters in SQL Functions 20-22
Linguistic Index Support 20-26
Importing and Loading Data Using NLS 20-27
Obtaining Information About NLS Settings 20-28
Summary 20-33

Appendix A: Practices

Environment A-2
Practice 1: Oracle Architectural Components A-3
Practice 2: Getting Started With Oracle A-5
Practice 3: Managing an Oracle Instance A-6
Practice 4: Creating a Database A-8
Practice 5: Creating Data Dictionary Views and Standard Packages A-9

Practice 6: Maintaining the Control File	A-10
Practice 7: Maintaining Redo Log Files	A-11
Practice 8: Managing Tablespaces and Data Files	A-12
Practice 9: Storage Structure and Relationships	A-13
Practice 10: Managing Rollback Segments	A-14
Practice 11: Managing Tables	A-15
Practice 12: Managing Indexes	A-17
Practice 13: Maintaining Data Integrity	A-19
Practice 14: Loading Data	A-20
Practice 15: Reorganizing Data	A-21
Practice 16: Managing Password Security	A-22
Practice 17: Managing Users	A-23
Practice 18: Managing Privileges	A-24
Practice 19: Managing Roles	A-25
Practice 20: Using National Language Support	A-26

Appendix B: Hints

Practice 1: Oracle Architectural Components	B-2
Practice 2: Getting Started With Oracle	B-3
Practice 3: Managing an Oracle Instance	B-4
Practice 4: Creating a Database	B-7
Practice 5: Creating Data Dictionary Views and Standard Packages	B-8
Practice 6: Maintaining the Control File	B-9
Practice 7: Maintaining Redo Log Files	B-10
Practice 8: Managing Tablespaces and Data Files	B-12
Practice 9: Storage Structure and Relationships	B-14
Practice 10: Managing Rollback Segments	B-16
Practice 11: Managing Tables	B-18
Practice 12: Managing Indexes	B-20
Practice 13: Maintaining Data Integrity	B-22
Practice 14: Loading Data	B-23
Practice 15: Reorganizing Data	B-25
Practice 16: Managing Password Security	B-26

- Practice 17: Managing Users B-27
- Practice 18: Managing Privileges B-28
- Practice 19: Managing Roles B-29
- Practice 20: Using National Language Support B-30

Appendix C: Practice Solutions for SQL*Plus

- Practice 1 Solutions C-2
- Practice 2 Solutions C-4
- Practice 3 Solutions C-7
- Practice 4 Solutions C-16
- Practice 5 Solutions C-20
- Practice 6 Solutions C-24
- Practice 7 Solutions C-27
- Practice 8 Solutions C-33
- Practice 9 Solutions C-40
- Practice 10 Solutions C-46
- Practice 11 Solutions C-54
- Practice 12 Solutions C-59
- Practice 13 Solution C-63
- Practice 14 Solutions C-69
- Practice 15 Solutions C-75
- Practice 16 Solutions C-80
- Practice 17 Solutions C-84
- Practice 18 Solutions C-87
- Practice 19 Solutions C-92
- Practice 20 Solutions C-95

Appendix D: Practice Solutions for Oracle Enterprise Manager

- Practice 1 Solutions D-2
- Practice 2 Solutions D-3
- Practice 3 Solutions D-4
- Practice 4 Solutions D-8
- Practice 5 Solutions D-9
- Practice 6 Solutions D-10

Practice 7 Solutions D-12
Practice 8 Solutions D-19
Practice 9 Solutions D-25
Practice 10 Solutions D-26
Practice 11 Solutions D-37
Practice 12 Solutions D-42
Practice 13 Solutions D-47
Practice 14 Solutions D-49
Practice 15 Solutions D-52
Practice 16 Solutions D-59
Practice 17 Solutions D-64
Practice 18 Solutions D-71
Practice 19 Solutions D-76
Practice 20 Solutions D-80

Appendix E: Certification Test: Sample Questions

Oracle Certified Professional (OCP) Program: Oracle Certified Database Administrator Track E-2
Oracle Database Administration: Sample Test E-3
Oracle Backup and Recovery Sample Test E-5
Answers E-8
Registering for an OCP Test E-9

Preface

Profile

This course is designed to give the Oracle database administrator (DBA) a firm foundation in basic administrative tasks. The primary goal of this course is to give the DBA the necessary knowledge and skills to set up, maintain, and troubleshoot an Oracle database. This course has been designed for database administrators, technical support analysts, system administrators, application developers, MIS managers, and other Oracle users.

This preface covers the following sections:

- Before You Begin This Course
- Prerequisites
- How This Course Is Organized
- How This Book Is Organized
- Related Publications
- Typographic Conventions

Before You Begin This Course

The specific skills you as a participant must have in order to derive the maximum value from attending this course are:

- Familiarity with relational database concepts
- Thorough knowledge of SQL, SQL*Plus, and PL/SQL
- Basic operating system knowledge
- Working experience with the Oracle environment

Prerequisites

- *SQL I*
- *PL/SQL Fundamentals*

How This Course Is Organized

Enterprise DBA Part 1A: Architecture and Administration is an instructor-led course featuring lectures and hands-on exercises. Online demonstrations, animation, and written practice sessions reinforce the concepts and skills introduced. The course also uses challenge-level practice labs, including scenarios and new drill-down topics for NT users.

In addition, bulletins from Oracle Worldwide Support that address the most frequently asked questions are used to prepare participants to troubleshoot real-world issues.

This course contains clearly defined objectives designed to support preparation for the *Oracle Certified Professional* examination.

Related Information

Oracle Publications

Title	Part Number
<i>Oracle8i Generic Documentation Set, Release 8.1.5</i>	A69148
<i>Oracle8i Enterprise Edition: Getting Started Release 8.1.5 for Windows NT</i>	A68694-02
<i>Oracle8i: Administrator's Reference Release 8.1.5 for Sun SPARC Solaris</i>	A67456-01
Oracle Press: <i>Oracle8: DBA Handbook</i>	A54760
Oracle Press: <i>Oracle8: A Beginner's Guide</i>	A54756
Oracle Press: <i>Oracle8: The Complete Reference</i>	A54759
<i>Oracle Enterprise Manager Documentation Set</i>	A67817

Web Sites

http://www.oracle.com
http://education.oracle.com
http://www.oramag.com

Oracle Education Student Union

In the Student Union (Oracle's online student community), you can continue to learn through online forums with Oracle instructors or online mini-lessons with streaming Real Video. You can also find valuable course information in the Resource Center.

Log in to the Oracle Education Student Union at <http://education.oracle.com>.

View one of the following mini-lessons:

- Oracle8i-Manageability Enhancements
- Overview of Oracle8i PL/SQL Features and Positioning

Additional Publications

- System release bulletins
- Installation and configuration guides
- International Oracle User's Group (IOUG) articles
- *Oracle Magazine*

Typographic Conventions

Typographic Conventions in Text

Convention	Element	Example
Bold italic	Glossary term (if there is a glossary)	The <i>algorithm</i> inserts the new key.
Caps and lowercase	Buttons, check boxes, triggers, windows	Click the Executable button. Select the Can't Delete Card check box. Assign a When-Validate-Item trigger . . . Open the Master Schedule window.
Courier new, case sensitive (default is lowercase)	Code output, directory names, filenames, passwords, pathnames, URLs, user input, usernames	Code output: <code>debug.seti('I',300);</code> Directory: <code>bin</code> (DOS), <code>\$FMHOME</code> (UNIX) Filename: Locate the <code>init.ora</code> file. Password: Use <code>tiger</code> as your password. Pathname: Open <code>c:\my_docs\projects</code> URL: Go to <code>http://www.oracle.com</code> User input: Enter <code>300</code> Username: Log on as <code>scott</code>
Initial cap	Graphics labels (unless the term is a proper noun)	Customer address (<i>but</i> Oracle Payables)
Italic	Emphasized words and phrases, titles of books and courses, variables	Do <i>not</i> save changes to the database. For further information, see <i>Oracle7 Server SQL Language Reference Manual</i> . Enter <i>user_id</i> @us.oracle.com, where <i>user_id</i> is the name of the user.
Quotation marks	Interface elements with long names that have only initial caps; lesson and chapter titles in cross-references	Select "Include a reusable module component" and click Finish. This subject is covered in Unit II, Lesson 3, "Working with Objects."
Uppercase	SQL column names, commands, functions, schemas, table names	Use the SELECT command to view information stored in the LAST_NAME column of the EMP table.

Convention	Element	Example
Arrow	Menu paths	Select File—>Save.
Brackets	Key names	Press [Enter].
Commas	Key sequences	Press and release these keys one at a time: [Alt], [F], [D]
Plus signs	Key combinations	Press and hold these keys simultaneously: [Ctrl]+[Alt]+[Del]

Typographic Conventions in Code

Convention	Element	Example
Caps and lowercase	Oracle Forms triggers	When-Validate-Item
Lowercase	Column names, table names	SELECT last_name FROM s_emp;
	Passwords	DROP USER scott IDENTIFIED BY tiger;
	PL/SQL objects	OG_ACTIVATE_LAYER (OG_GET_LAYER ('prod_pie_layer'))
Lowercase italic	Syntax variables	CREATE ROLE <i>role</i>
Uppercase	SQL commands and functions	SELECT userid FROM emp;

Typographic Conventions in Navigation Paths

This course uses simplified navigation paths, such as the following example, to direct you through Oracle Applications.

(N) Invoice—>Entry—>Invoice Batches Summary (M) Query—>Find
(B) Approve

This simplified path translates to the following:

- 1 (N) From the Navigator window, select Invoice—>Entry—>Invoice Batches Summary.
- 2 (M) From the menu bar, select Query—>Find.
- 3 (B) Click the Approve button.

N = Navigator, **M** = Menu, **B** = Button

Introduction

Objectives

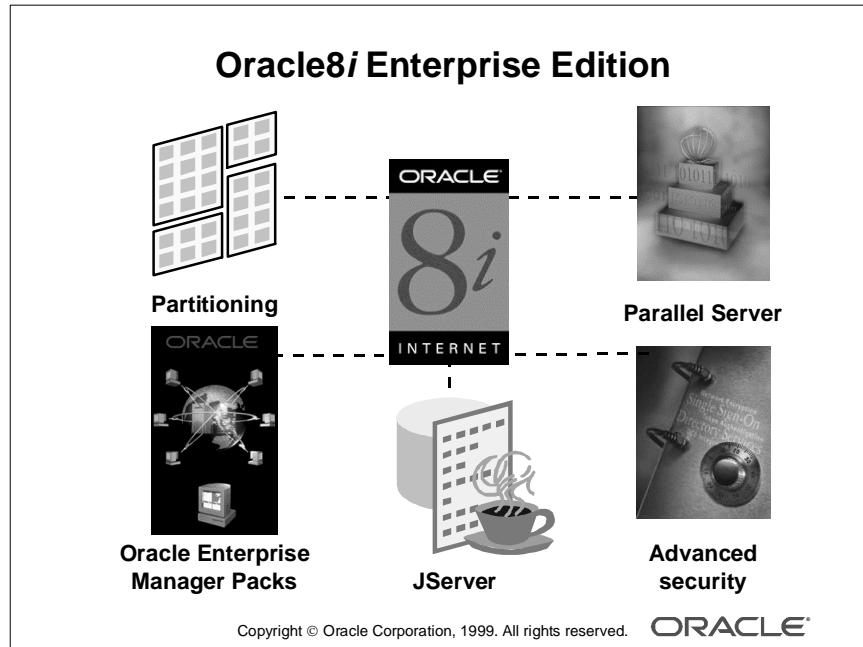
Course Objectives

After completing this course, you should be able to do the following:

- **Start up and shut down an Oracle instance and database**
- **Create an operational database**
- **Manage Oracle database files**
- **Manage tablespaces, segments, extents, and blocks**
- **Manage users, privileges, and resources**
- **Use National Language Support (NLS) features**

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

Oracle8i Enterprise Edition



What Is the Oracle8i Enterprise Edition?

Oracle8i Enterprise Edition is an object relational database that is scalable and easily manageable. The administration of the basic enterprise edition is discussed in this course. However, the following options provide additional functionality:

- **Partitioning:** Provides facilities for implementing large, scalable applications. (It enables control over tables and indexes at a lower level of granularity than is possible with the basic enterprise edition.)
- **Oracle Parallel Server:** Improves the scalability and availability of a database by allowing multiple copies of the Oracle software to access a single database.
- **Oracle Enterprise Manager Packs:** Built on top of the Oracle Enterprise Manager. Oracle Enterprise Manager Diagnostics, Tuning, and Change Management Packs are add-ons that provide DBAs with a set of tools for advanced diagnostics, monitoring, tuning, and change management of Oracle environments. Oracle Enterprise Manager also provides a DBA Management Pack that enables DBAs to do basic administration, such as creating users, starting up an instance, granting privileges, and so on.

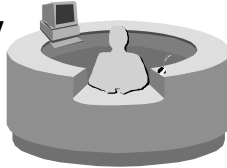
What Is the Oracle8i Enterprise Edition? (continued)

- Oracle JServer: Enables users to define a variety of application programming models, including Java stored procedures and triggers, Java methods of object relational types, CORBA objects, Enterprise JavaBeans, Java servlets, and Java Server Pages. It also supports a variety of standards-based protocols such as Internet Inter-ORB Protocol (IIOP) and Hypertext Transfer Protocol (HTTP), in addition to Net8.
- Advanced security: Provides client-server, server-server network security, using encryption and data integrity checking, and supports enhanced user authentication services, using third-party security services.

Database Administrator Tasks

Database Administrator Tasks

- Manage database availability
- Plan and create databases
- Manage physical structures
- Manage storage based on design
- Manage security



- Network administration
- Backup and recovery
- Database tuning

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE™

Scope of the Course

This course is the first in a series of four courses that cover the core database administrator tasks. The tasks covered in this course are:

- Managing database availability
- Planning and creating databases
- Managing the physical structures
- Managing storage, based on the design specifications
- Managing database users, and controlling and monitoring their actions

DBA Tasks Covered in Other Courses

The following tasks are discussed in other courses:

- Network administration in *Enterprise DBA Part 3: Network Administration*
- Backup and recovery in *Enterprise DBA Part 1B: Backup and Recovery Workshop*
- Database tuning in *Enterprise DBA Part 2: Performance Tuning Workshop*
- Oracle Enterprise Manager in *Oracle Enterprise Manager V2*

Course Schedule

Suggested Course Schedule

Day	Start	End
1	Lesson 1	Lesson 4
2	Lesson 4 (Practice)	Lesson 7
3	Lesson 8	Lesson 11
4	Lesson 12	Lesson 15
5	Lesson 16	Lesson 20

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE[®]

Course Schedule

The following is the recommended lesson schedule for this course:

Day 1

Lesson 1: Oracle Architectural Components

Lesson 2: Getting Started with the Oracle Server

Lesson 3: Managing an Oracle Instance

Lesson 4: Creating a Database (Lecture)

Day 2

Lesson 4: Creating a Database (Practice)

Lesson 5: Creating Data Dictionary Views and Standard Packages

Lesson 6: Maintaining the Control File

Lesson 7: Maintaining Redo Log Files

Course Schedule (continued)

Day 3

Lesson 8: Managing Tablespaces and Data Files

Lesson 9: Storage Structure and Relationships

Lesson 10: Managing Rollback Segments

Lesson 11: Managing Tables

Day 4

Lesson 12: Managing Indexes

Lesson 13: Maintaining Data Integrity

Lesson 14: Loading Data

Lesson 15: Reorganizing Data

Day 5

Lesson 16: Managing Password Security and Resources

Lesson 17: Managing Users

Lesson 18: Managing Privileges

Lesson 19: Managing Roles

Lesson 20: Using National Language Support

Oracle Architectural Components

Objectives

Objectives

After completing this lesson, you should be able to do the following:

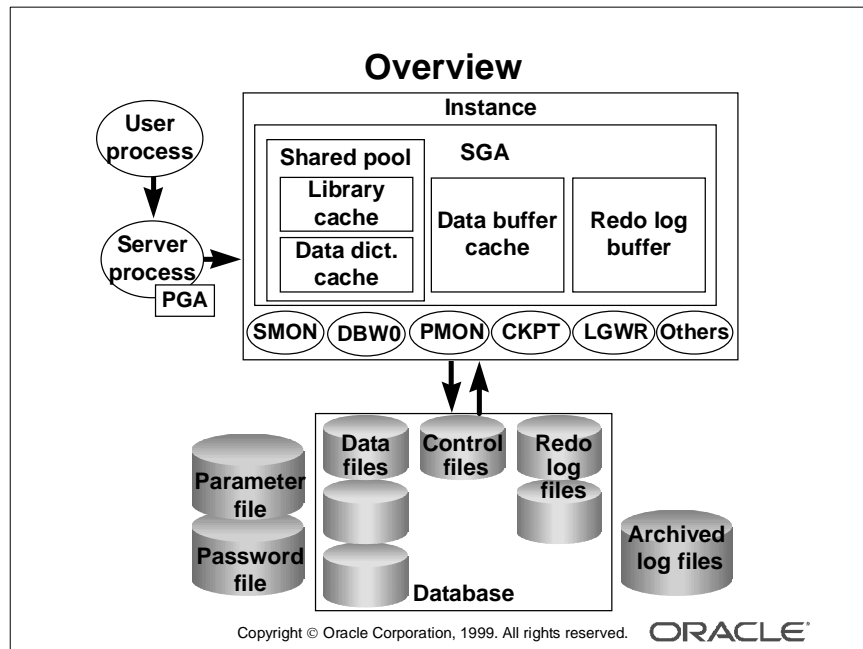
- **Describe the Oracle server architecture and its main components**
- **List the structures involved in connecting a user to an Oracle instance**
- **List the stages in processing:**
 - **Queries**
 - **DML statements**
 - **COMMITs**

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE

Objectives

This lesson introduces the Oracle server architecture by examining the files, processes, and memory structures involved in establishing a database connection and executing a SQL command.

Overview



Overview

The Oracle server is an object relational database management system that provides an open, comprehensive, integrated approach to information management.

Primary Components

There are several processes, memory structures, and files in an Oracle server; however, not all of them are used when processing a SQL statement. Some are used to improve the performance of the database, ensure that the database can be recovered in the event of a software or hardware error, or perform other tasks necessary to maintain the database. The Oracle server consists of an Oracle instance and an Oracle database.

Oracle Instance An Oracle instance is the combination of the background processes and memory structures. The instance must be started to access the data in the database. Every time an instance is started, a System Global Area (SGA) is allocated and Oracle background processes are started.

- The SGA is a memory area used to store database information that is shared by database processes.

Primary Components (continued)

Oracle Instance (continued)

- Background processes perform functions on behalf of the invoking process. They consolidate functions that would otherwise be handled by multiple Oracle programs running for each user. The background processes perform I/O and monitor other Oracle processes to provide increased parallelism for better performance and reliability.

Other Processes The user process is the application program that originates SQL statements. The server process executes the SQL statements sent from the user process.

Database Files Database files are operating system files that provide the actual physical storage for database information. The database files are used to ensure that the data is kept consistent and can be recovered in the event of a failure of the instance.

Other Files Nondatabase files are used to configure the instance, authenticate privileged users, and recover the database in the event of a disk failure.

SQL Statement Processing

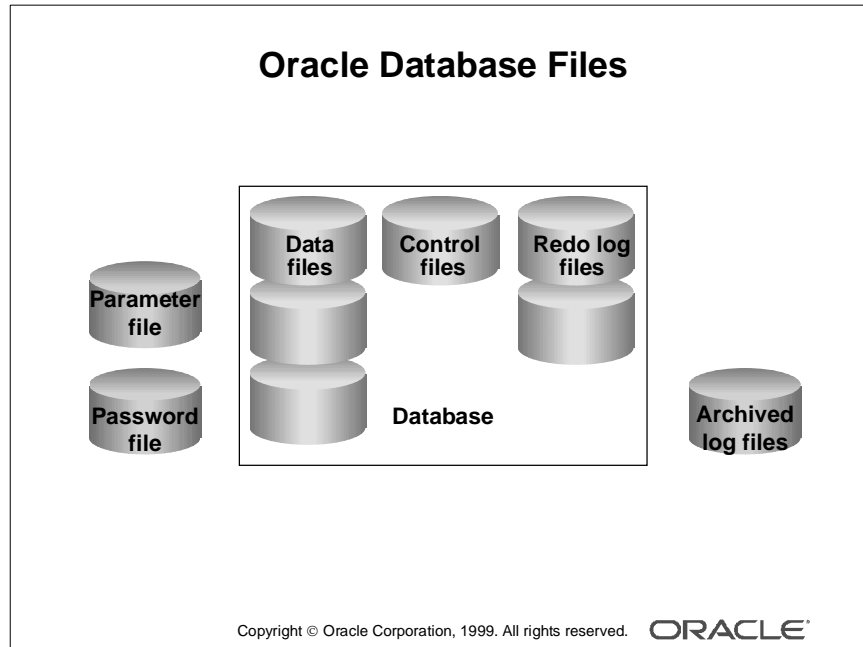
The user and server processes are the primary processes involved when a SQL statement is executed; however, other processes may help the server complete the processing of the SQL statement.

Oracle Database Administrators

Database administrators are responsible for maintaining the Oracle server so that the server can process user requests. An understanding of the Oracle architecture is necessary to maintain it effectively.

This course focuses on creating and maintaining an Oracle server where users connect either by directly logging in to the machine running the Oracle server or by using a client-server model.

Oracle Database Files

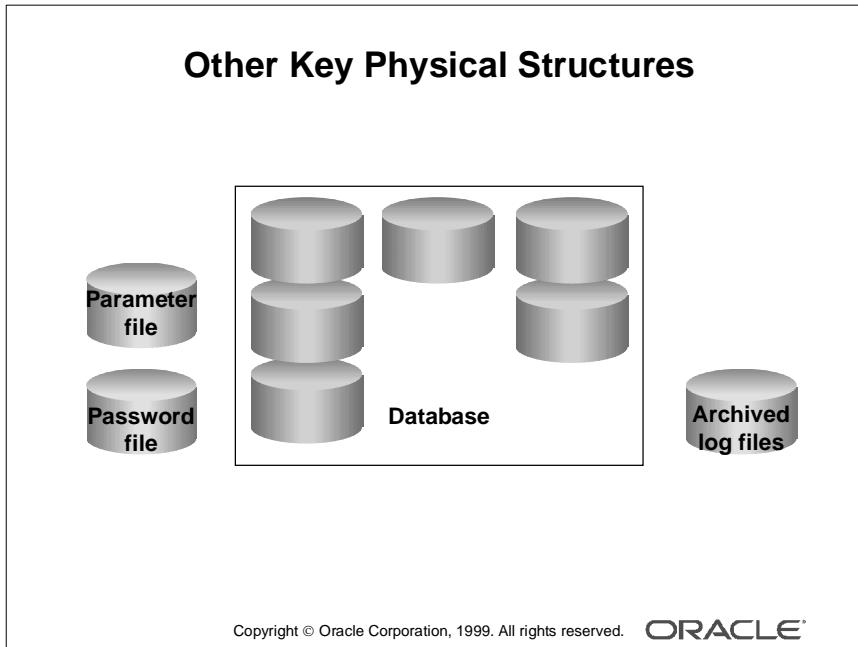


Oracle Database Files

An Oracle database is a collection of data that is treated as a unit. The general purpose of a database is to store and retrieve related information. The database has a logical structure and a physical structure. The physical structure of the database is the set of operating system files in the database. An Oracle database consists of three file types:

- Data files contain the actual data in the database. The data is stored in user-defined tables, but data files also contain the data dictionary, before-images of modified data, indexes, and other types of structures. A database has at least one data file. The characteristics of data files are:
 - A data file can be associated with only one database.
 - Data files can have certain characteristics set to allow them to automatically extend when the database runs out of space.
 - One or more data files form a logical unit of database storage called a tablespace. The logical structure of the database is discussed in Lesson 8, “Managing Tablespaces and Data Files.”
- Redo logs contain a record of changes made to the database to enable recovery of the data in case of failures. A database requires at least two redo log files.
- Control files contain information necessary to maintain and verify database integrity. For example, a control file is used to identify the data files and redo log files. A database needs at least one control file.

Other Key Files

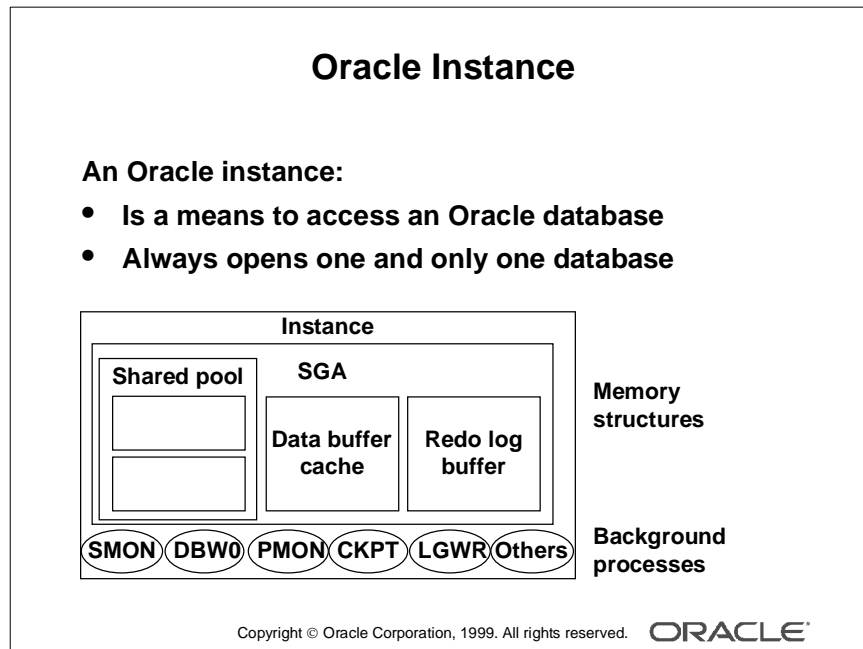


Other Key Files

The Oracle server also uses other files that are not part of the database:

- The parameter file defines the characteristics of an Oracle instance. For example, it contains parameters that size some of the memory structures in the SGA.
- The password file authenticates which users are permitted to start up and shut down an Oracle instance.
- Archived redo log files are offline copies of the redo log files that may be necessary to recover from media failures.

Oracle Instance



Oracle Instance

An Oracle instance consists of the SGA memory structure and the background processes used to manage a database. An instance is identified by using methods specific to each operating system. The instance can open and use only one database at a time.

System Global Area

The SGA is a memory area used to store database information that is shared by database processes. It contains data and control information for the Oracle server. It is allocated in the virtual memory of the computer where the Oracle server resides. The SGA consists of several memory structures:

- The shared pool is used to store the most recently executed SQL statements and the most recently used data from the data dictionary. These SQL statements may be submitted by a user process or, in the case of stored procedures, read from the data dictionary.
- The database buffer cache is used to store the most recently used data. The data is read from, and written to, the data files.
- The redo log buffer is used to track changes made to the database by the server and background processes.

System Global Area (continued)

The purpose of these structures is discussed in detail in later sections of this lesson.

There are also two optional memory structures in the SGA:

- Java pool: Used to store Java code
- Large pool: Used to store large memory structures not directly related to SQL statement processing; for example, data blocks copied during backup and restore operations

Background Processes

The background processes in an instance perform common functions that are needed to service requests from concurrent users without compromising the integrity and performance of the system. They consolidate functions that would otherwise be handled by multiple Oracle programs running for each user. The background processes perform I/O and monitor other Oracle processes to provide increased parallelism for better performance and reliability.

Depending on its configuration, an Oracle instance may include several background processes, but every instance includes these five required background processes:

- Database Writer (DBW0) is responsible for writing changed data from the database buffer cache to the data files.
- Log Writer (LGWR) writes changes registered in the redo log buffer to the redo log files.
- System Monitor (SMON) checks for consistency of the database and, if necessary, initiates recovery of the database when the database is opened.
- Process Monitor (PMON) cleans up resources if one of the Oracle processes fails.
- The Checkpoint Process (CKPT) is responsible for updating database status information in the control files and data files whenever changes in the buffer cache are permanently recorded in the database.

The following sections of this lesson explain how a server process uses some of the components of the Oracle instance and database to process SQL statements submitted by a user process.

Processing a SQL Statement

Processing a SQL Statement

- **Connect to an instance using:**
 - The user process
 - The server process
- **The Oracle server components that are used depend on the type of SQL statement:**
 - Queries return rows.
 - DML statements log changes.
 - Commit ensures transaction recovery.
- **Some Oracle server components do not participate in SQL statement processing.**

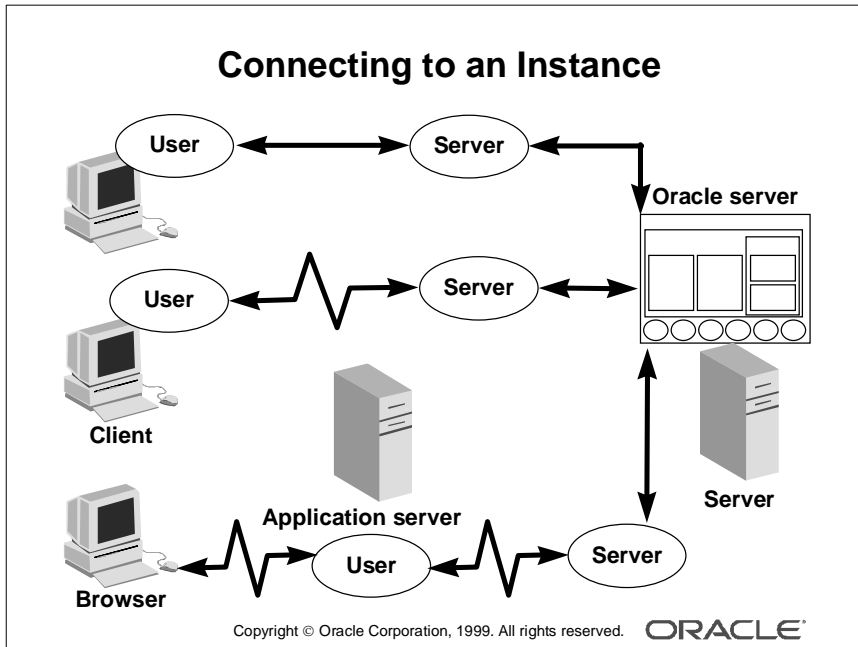
Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE

Components Used to Process SQL

Not all of the components of an Oracle instance are used to process SQL statements.

- The user and server process are used to connect a user to an Oracle instance. These processes are not part of the Oracle instance, but are required to process a SQL statement.
- Some of the background processes, SGA structures, and database files are used to process SQL statements. Depending on the type of SQL statement, different components are used:
 - Queries require additional processing to return rows to the user.
 - Data manipulation language (DML) statements require additional processing to log the changes made to the data.
 - Commit processing ensures that the modified data in a transaction can be recovered.
- Some required background processes do not directly participate in processing a SQL statement, but are used to improve performance and to recover the database.
- The optional background process, ARC0, is used to ensure that a production database can be recovered.

Connecting to a Database



Processes Used to Connect to an Instance

Before users can submit SQL statements to Oracle, they must connect to an instance.

- The user starts a tool such as SQL*Plus or runs an application developed using a tool such as Oracle Forms. This application or tool is executed in a *user process*.
- In the most basic configuration, when a user logs on to the Oracle server, a process is created on the computer running the Oracle server. This process is called a server process. The server process communicates with the Oracle instance on behalf of the user process that runs on the client. The server process executes SQL statements on behalf of the user.

Connection

A connection is a communication pathway between a user process and an Oracle server. A database user can connect to an Oracle server in one of three ways:

- The user logs on to the operating system running the Oracle instance and starts an application or tool that accesses the database on that system. The communication pathway is established using the interprocess communication mechanisms available on the host operating system.

Connection (continued)

- The user starts the application or tool on a local computer and connects over a network to the computer running the Oracle instance. In this configuration, called client-server, network software is used to communicate between the user and the Oracle server.
- In a three-tiered connection, the user's computer communicates over the network to an application or a network server, which is connected through a network to the machine running the Oracle instance. For example, the user runs a browser on a network computer to use an application residing on an NT server that retrieves data from an Oracle database running on a UNIX host.

Sessions

A session is a specific connection of a user to an Oracle server. The session starts when the user is validated by the Oracle server, and it ends when the user logs out or when there is an abnormal termination. For a given database user, many concurrent sessions are possible if the user logs on from many tools, applications, or terminals at the same time. Except for some specialized database administration tools, starting a database session requires that the Oracle server be available for use.

Note: The type of connection explained here, where there is a one-to-one correspondence between a user and server process, is called a dedicated server connection. When using a multithreaded server (MTS) configuration, it is possible for multiple user processes to share server processes. MTS is covered in more detail in the course *Enterprise DBA Part 3: Network Administration*.

Processing a Query

Processing a Query

- **Parse:**
 - Search for identical statement
 - Check syntax, object names, and privileges
 - Lock objects used during parse
 - Create and store execution plan
- **Execute:** Identify rows selected
- **Fetch:** Return rows to user process

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE

Query Processing Steps

Queries are different from other types of SQL statements because, if successful, they return data as results. Whereas other statements simply return success or failure, a query can return one row or thousands of rows.

There are three main stages in the processing of a query:

- 1 Parse
- 2 Execute
- 3 Fetch

Parsing a SQL Statement During the *parse* stage, the SQL statement is passed from the user process to the server process, and a parsed representation of the SQL statement is loaded into a shared SQL area.

During the parse, the server process:

- Searches for an existing copy of the SQL statement in the shared pool
- Validates the SQL statement by checking its syntax
- Performs data dictionary lookups to validate table and column definitions

Query Processing Steps (continued)

Parsing a SQL Statement (continued)

- Acquires parse locks on objects so that their definitions do not change during the parsing of the statement
- Checks the user's privileges to access the referenced schema objects
- Determines the optimal execution plan for the statement
- Loads the statement and execution plan into a shared SQL area

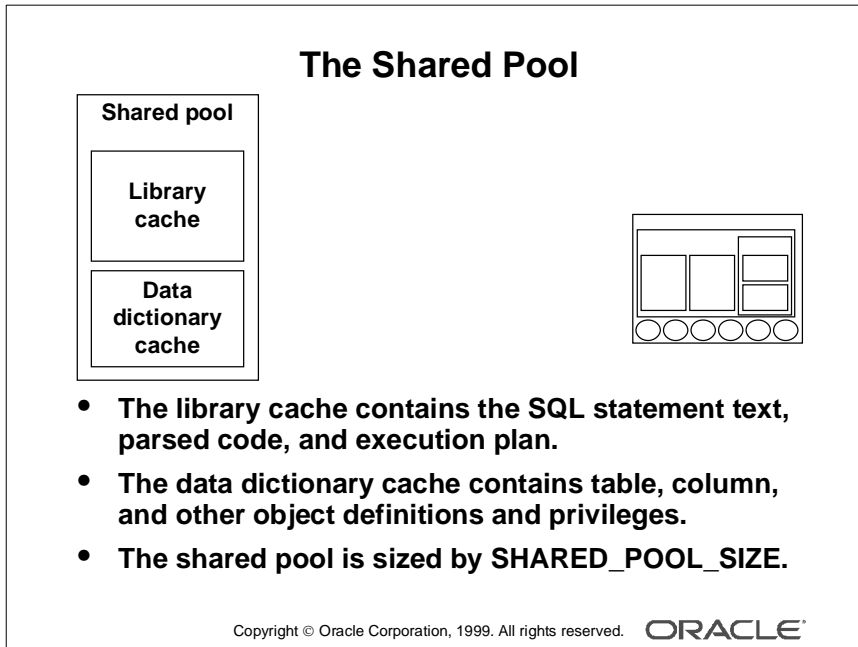
The parse stage includes processing requirements that usually need to be done only once no matter how many times the statement is executed. The Oracle server usually translates each SQL statement only once, reexecuting that parsed statement during subsequent references to the statement. However, the Oracle server always verifies that the user has the required privileges to execute the SQL statement.

Although parsing a SQL statement validates that statement, parsing only identifies errors that can be found before statement execution. Thus, some errors cannot be caught by parsing. For example, errors in data conversion or errors in data (such as an attempt to enter duplicate values in a primary key) and deadlocks are all errors or situations that can be encountered and reported only during the execution stage.

Executing the SELECT Statement At this point, the Oracle server has all the necessary information and resources, so the statement is executed. For SELECT statements, the server process prepares to retrieve the data.

Fetching the Rows of a Query In the fetch stage, rows are selected, ordered (if necessary), and returned by the server to the user. Depending on the number of rows returned in each fetch, one or more fetches may be required to transfer the results of a query to the user.

The Shared Pool



Shared Pool Components

During the parse stage, the server process uses the area in the SGA known as the shared pool to compile the SQL statement. The shared pool has two primary components:

- Library cache
- Data dictionary cache

Library Cache The library cache stores information about the most recently used SQL statements in a memory structure called a shared SQL area. The shared SQL area contains:

- The text of the SQL statement
- The parse tree: A compiled version of the statement
- The execution plan: The steps to be taken when executing the statement

The optimizer is the function in the Oracle server that determines the optimal execution plan.

Shared Pool Components (continued)

Library Cache (continued) If a SQL statement is reexecuted and a shared SQL area already contains the execution plan for the statement, the server process does not need to parse the statement. The library cache improves the performance of applications that reuse SQL statements by reducing parse time and memory requirements. If the SQL statement is not reused, it is eventually aged out of the library cache.

Data Dictionary Cache

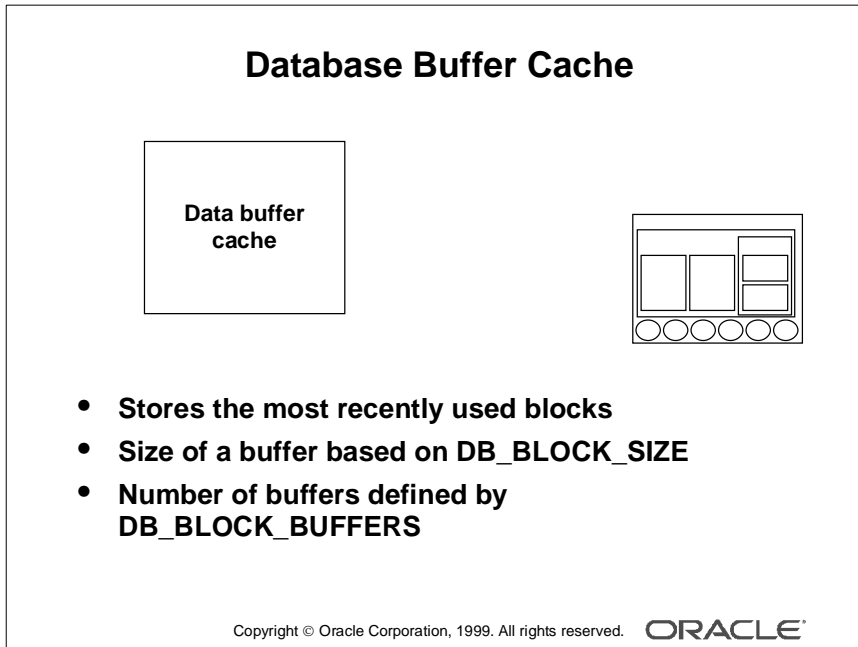
The data dictionary cache, also known as the dictionary cache or row cache, is a collection of the most recently used definitions in the database. It includes information about database files, tables, indexes, columns, users, privileges, and other database objects.

During the parse phase, the server process looks for the information in the dictionary cache to resolve the object names specified in the SQL statement and to validate the access privileges. If necessary, the server process initiates the loading of this information from the data files.

Sizing the Shared Pool

The size of the shared pool is specified by the initialization parameter `SHARED_POOL_SIZE`.

Database Buffer Cache



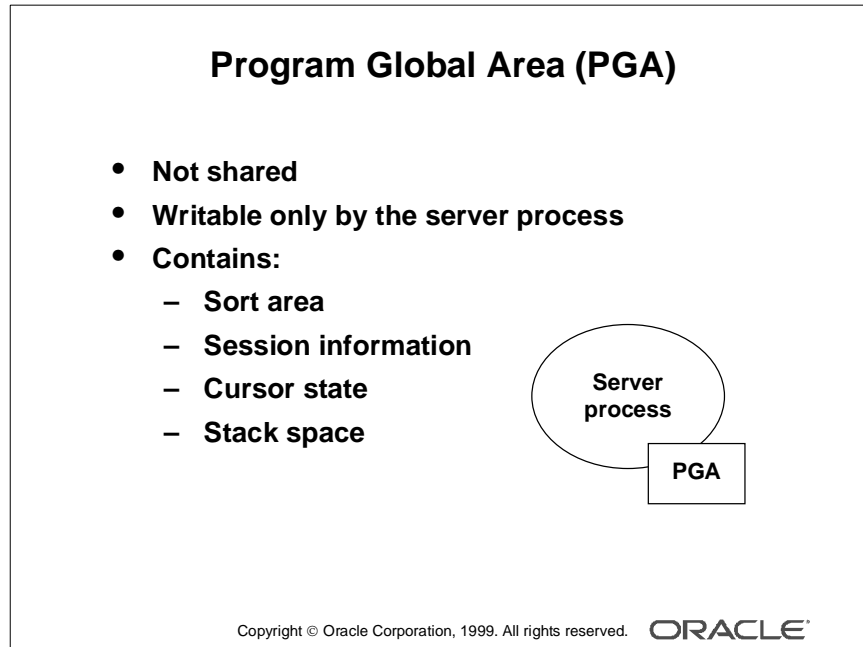
Function of the Database Buffer Cache

When a query is processed, the server process looks in the database buffer cache for any blocks it needs. If the block is not found in the database buffer cache, the server process reads the block from the data file and places a copy in the buffer cache. Because subsequent requests for the same block may find the block in memory, the requests may not require physical reads. The Oracle server uses a least recently used algorithm to age out buffers that have not been accessed recently to make room for new blocks in the buffer cache.

Sizing the Database Buffer Cache

The size of each buffer in the buffer cache is equal to the size of an Oracle block, and it is specified by the DB_BLOCK_SIZE parameter. The number of buffers is equal to the value of the DB_BLOCK_BUFFERS parameter.

Program Global Area



Program Global Area Components

The Program Global Area or Process Global Area (PGA) is a memory region that contains data and controls information for a single server process or a single background process. In contrast to the SGA, which is shared by several processes, the PGA is an area that is used by only one process. In a dedicated server configuration, the PGA of the server includes these components:

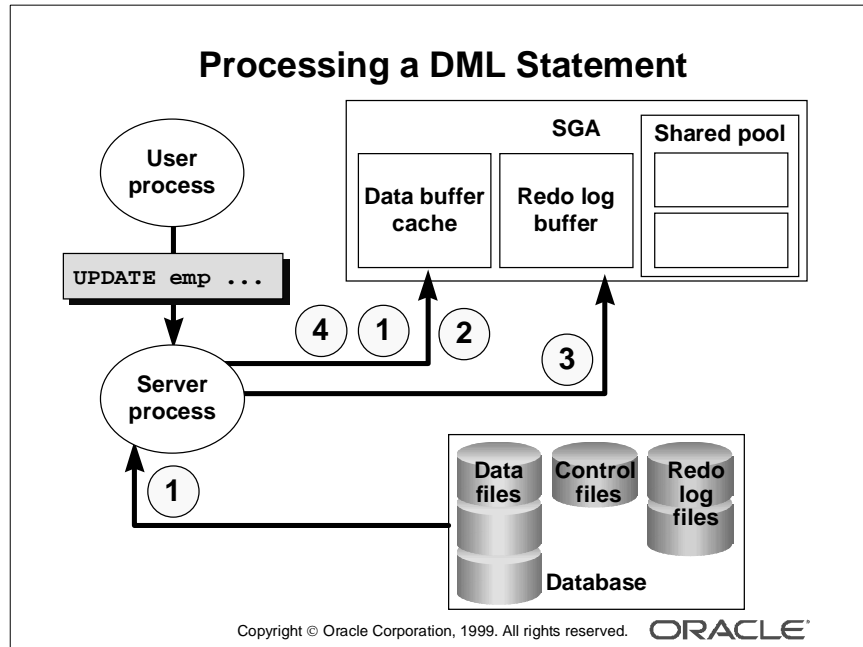
- **Sort area:** Used for any sorts that may be required to process the SQL statement
- **Session information:** Includes user privileges and performance statistics for the session
- **Cursor state:** Indicates the stage in the processing of the SQL statements that are currently used by the session
- **Stack space:** Contains other session variables

The PGA is allocated when a process is created and deallocated when the process is terminated.

Program Global Area (continued)

Note: Some of these structures are stored in the SGA when using a multithreaded server (MTS) configuration. With an MTS configuration, it is possible for multiple user processes to share server processes. MTS is covered in more detail in the course *Enterprise DBA Part 3: Network Administration*. If a large pool is created, the structures are stored there; otherwise, they are stored in the shared pool.

Processing a DML Statement



DML Processing Steps

A data manipulation language (DML) statement requires only two phases of processing:

- Parse is the same as the parse phase used for processing a query
- Execute requires additional processing to make data changes

DML Execute Phase

To execute a DML statement:

- 1 If the data and rollback blocks are not already in the buffer cache, the server process reads them from the data files into the buffer cache.
- 2 The server process places locks on the rows that are to be modified.
- 3 In the redo log buffer, the server process records the changes to be made to the rollback and data.
 - The rollback block changes record the values of the data before it is modified. The rollback block is used to store the before-image of the data, so that the DML statements can be rolled back if necessary.
 - The data blocks changes record the new values of the data.

DML Execute Phase (continued)

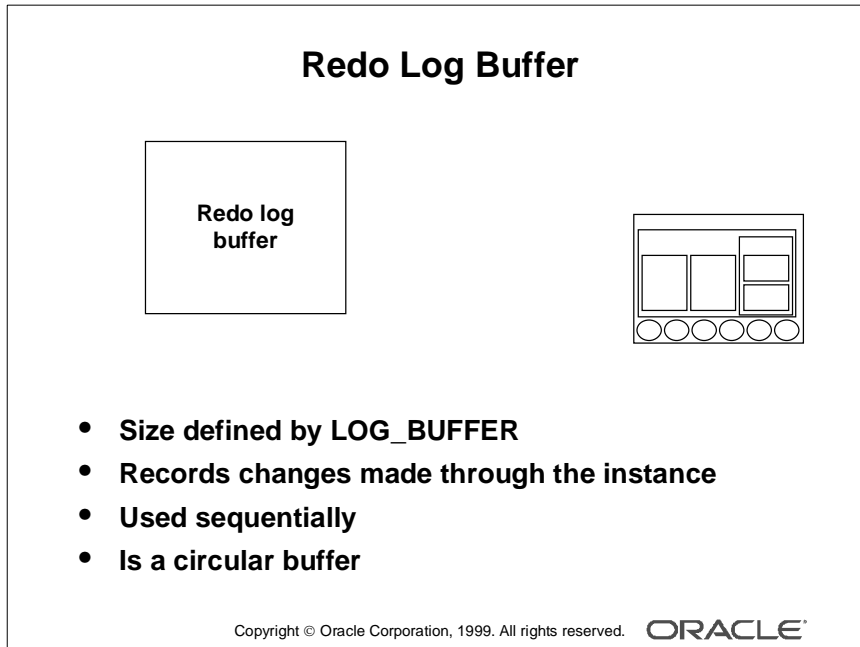
- 4 The server process records the before-image to the rollback block and updates the data block. Both of these changes are done in the database buffer cache. Any changed blocks in the buffer cache are marked as dirty buffers—that is, buffers that are not the same as the corresponding blocks on the disk.

The processing of a DELETE or INSERT command uses similar steps. The before-image for a DELETE contains the column values in the deleted row, and the before-image of an INSERT contains the row location information.

Technical Note

Because the changes made to the blocks are only recorded in memory structures and are not written immediately to disk, a computer failure that causes the loss of the SGA may also lose these changes.

Redo Log Buffer



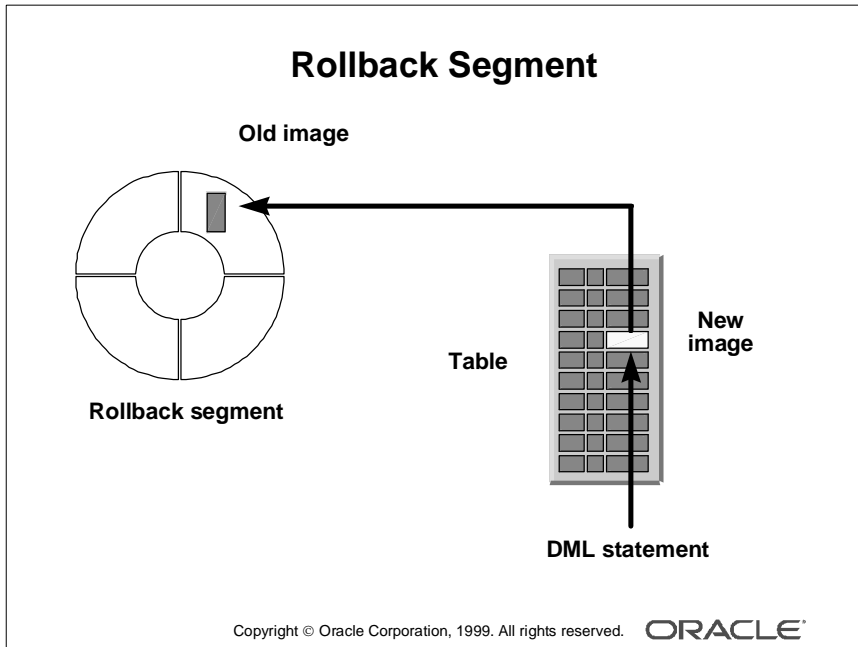
Redo Log Buffer Characteristics

The server process records most of the changes made to data file blocks in the redo log buffer, which is a part of the SGA. The redo log buffer has the following characteristics:

- Its size in bytes is defined by the LOG_BUFFER parameter.
- It records the block that is changed, the location of the change, and the new value in a redo entry. A redo entry makes no distinction between the type of block that is changed; it simply records which bytes are changed in the block.
- The redo log buffer is used sequentially, and changes made by one transaction may be interleaved with changes made by other transactions.
- It is a circular buffer that is reused after it is filled, but only after all the old redo entries are recorded in the redo log files.

Note: The redo log files are covered in more detail in the lesson “Maintaining Redo Log Files.”

Rollback Segment



Rollback Segment

Before making a change, the server process saves the old data value into a rollback segment. This before-image is used to:

- Undo the changes if the transaction is rolled back
- Provide read consistency by ensuring that other transactions do not see uncommitted changes made by the DML statement
- Recover the database to a consistent state in case of failures

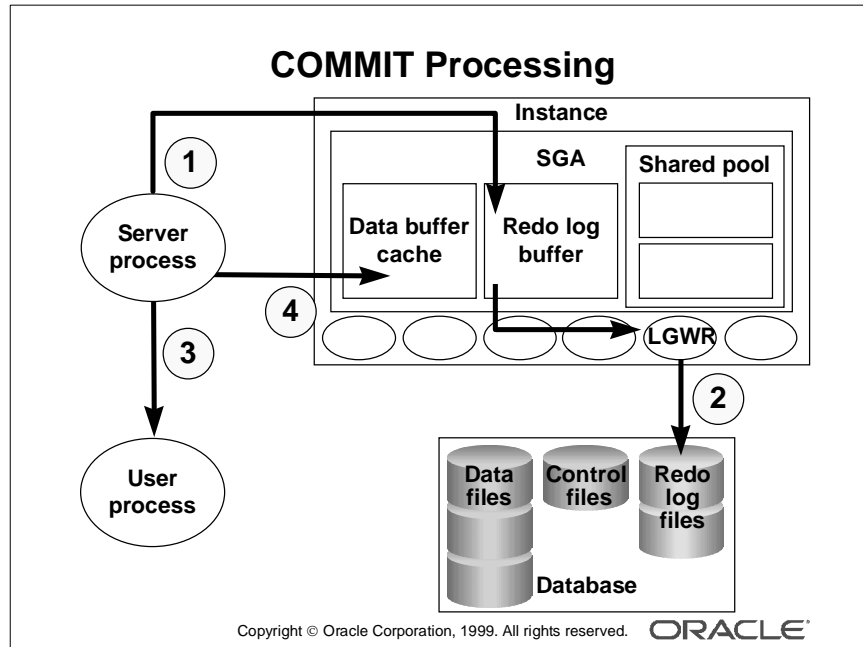
Rollback segments, like tables and indexes, exist in data files, and rollback blocks are brought into the database buffer cache as required.

Rollback segments are created by the DBA.

Changes to rollback segments are recorded in the redo log buffer.

More information about rollback segments is provided in the lesson “Managing Rollback Segments.”

COMMIT Processing



Fast COMMIT

The Oracle server uses a fast commit mechanism that guarantees that the committed changes can be recovered in case of instance failure.

System Change Number

Whenever a transaction commits, the Oracle server assigns a commit system change number (SCN) to the transaction. The SCN is monotonically incremented and is unique within the database. It is used by the Oracle server as an internal time stamp to synchronize data and to provide read consistency when data is retrieved from the data files. Using the SCN enables the Oracle server to perform consistency checks without depending on the date and time of the operating system.

Steps in Processing COMMITs

When a COMMIT is issued, the following steps are performed:

- 1 The server process places a commit record, along with the SCN, in the redo log buffer.
- 2 LGWR performs a contiguous write of all the redo log buffer entries up to and including the commit record to the redo log files. After this point, the Oracle server can guarantee that the changes will not be lost even if there is an instance failure.

Steps in Processing COMMITs (continued)

- 3 The user is informed that the COMMIT is complete.
- 4 The server process records information to indicate that the transaction is complete and that resource locks can be released.

Flushing of the dirty buffers to the data file is performed independently by DBW0 and can occur either before or after the commit.

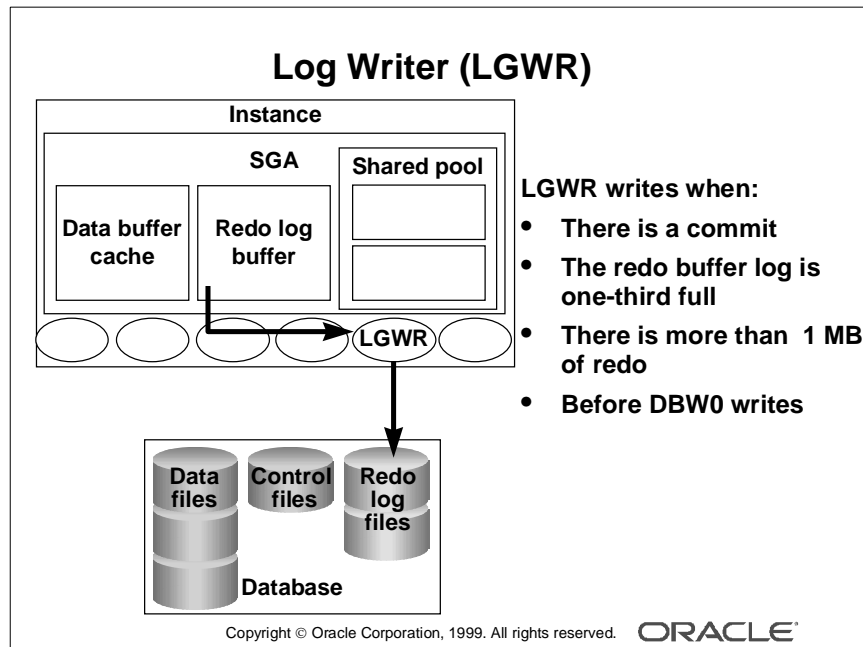
Advantages of the Fast COMMIT

The fast commit mechanism ensures data recovery by writing changes to the redo log buffer instead of the data files. It has the following advantages:

- Sequential writes to the log files are faster than writing to different blocks in the data file.
- Only the minimal information that is necessary to record changes is written to the log files, whereas writing to the data files would require whole blocks of data to be written.
- If multiple transactions request to commit at the same time, the instance piggybacks redo log records into a single write.
- Unless the redo log buffer is particularly full, only one synchronous write is required per transaction. If piggybacking occurs, there can be less than one synchronous write per transaction.
- Because the redo log buffer may be flushed before the COMMIT, the size of the transaction does not affect the amount of time needed for an actual COMMIT operation.

Note: Rolling back a transaction does not trigger LGWR to write to disk. The Oracle server always rolls back uncommitted changes when recovering from failures. If there is a failure after a rollback, before the rollback entries are recorded on disk, the absence of a commit record is sufficient to ensure that the changes made by the transaction are rolled back.

Log Writer



LOG Writer

LGWR performs sequential writes from the redo log buffer to the redo log file under the following situations:

- When a transaction commits
- When the redo log buffer is one-third full
- When there is more than a megabyte of changes recorded in the redo log buffer
- Before DBW0 writes modified blocks in the database buffer cache to the data files

Because the redo is needed for recovery, LGWR confirms the COMMIT only after the redo is written to disk.

Other Instance Processes

Other Instance Processes

- **Other required processes:**
 - **Database Writer (DBW0)**
 - **Process Monitor (PMON)**
 - **System Monitor (SMON)**
 - **Checkpoint (CKPT)**
- **The archive process (ARC0) is usually created in a production database**

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

Other Required Processes

Four other required processes do not participate directly in processing SQL statements:

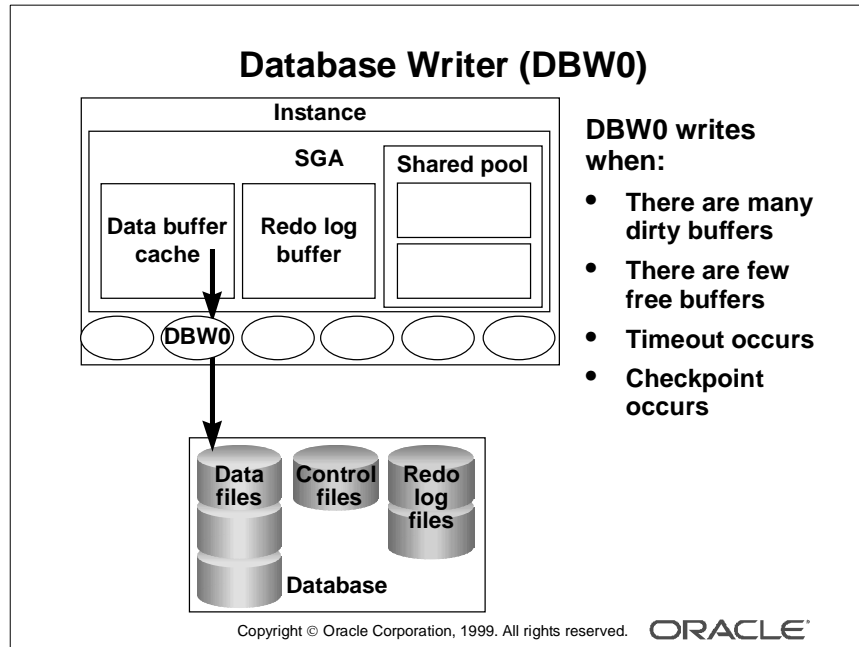
- Database Writer (DBW0)
- Process Monitor (PMON)
- System Monitor (SMON)
- Checkpoint (CKPT)

The checkpoint process is used to synchronize database files. It is covered in detail in the lesson “Maintaining Redo Log Files.”

The Archiver Process

All other background processes are optional, depending on the configuration of the database; however, one of them, ARC0, is crucial to recovering a database after the loss of a disk. The ARC0 process is usually created in a production database.

Database Writer



Database Writer

The server process records changes to rollback and data blocks in the buffer cache. The Database Writer (DBW0) writes the dirty buffers from the database buffer cache to the data files. It ensures that a sufficient number of free buffers—buffers that can be overwritten when server processes need to read in blocks from the data files—are available in the database buffer cache. Database performance is improved because server processes make changes only in the buffer cache, and the DBW0 defers writing to the data files until one of the following events occurs:

- The number of dirty buffers reaches a threshold value
- A process scans a specified number of blocks when scanning for free buffers and cannot find any
- A timeout occurs (every three seconds)
- A checkpoint occurs (A checkpoint is a means of synchronizing the database buffer cache with the data file. Checkpoints are covered in detail in the lesson “Maintaining Redo Log Files.”)

SMON: System Monitor

SMON: System Monitor

- **Automatically recovers the instance**
 - **Rolls forward changes in the redo logs**
 - **Opens the database for user access**
 - **Rolls back uncommitted transactions**
- **Coalesces free space**
- **Deallocates temporary segments**

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE

SMON: System Monitor

If the Oracle instance fails, any information in the SGA that has not been written to disk is lost. For example, the failure of the operating system causes an instance failure. After the loss of the instance, the background process SMON automatically performs instance recovery when the database is reopened. Instance recovery consists of the following steps:

- 1** Rolling forward to recover data that has not been recorded in the data files but that has been recorded in the online redo log. This data has not been written to disk because of the loss of the SGA during instance failure. During this process, SMON reads the redo log files and applies the changes recorded in the redo log to the data blocks. Because all committed transactions have been written to the redo logs, this process completely recovers these transactions.
- 2** Opening the database to allow users to log on. Any data that is not locked by unrecovered transactions is immediately available.
- 3** Rolling back uncommitted transactions. They are rolled back by SMON or by the individual server processes as they access locked data.

SMON: System Monitor (continued)

SMON also performs some space maintenance functions:

- It combines, or coalesces, adjacent areas of free space in the data files.
- It deallocates temporary segments to return them as free space in data files.
Temporary segments are used to store data during SQL statement processing.

PMON: Process Monitor

PMON: Process Monitor

Cleans up after failed processes by:

- **Rolling back the transaction**
- **Releasing locks**
- **Releasing other resources**

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE

PMON Functionality

The background process PMON cleans up after failed processes by:

- Rolling back the user's current transaction
- Releasing all currently held table or row locks
- Freeing other resources currently reserved by the user

Archiving

Archiving

- **Database archive mode**
 - **NOARCHIVELOG** for DBs that do not require recovery after a disk failure
 - **ARCHIVELOG** mode for production
- **ARC0 process**
 - **Automatically archives online redo logs**
 - **Preserves the record of all changes made to the database**

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

The Archiver Process

All other background processes are optional, depending on the configuration of the database; however, one of them, ARC0, is crucial to recovering a database after the loss of a disk. As online redo log files fill, the Oracle server begins writing to the next online redo log file. The process of switching from one redo log to another is called a log switch.

Archiving Redo Log Files

One of the important decisions that a DBA has to make is whether to configure the database to operate in ARCHIVELOG or in NOARCHIVELOG mode.

NOARCHIVELOG Mode In NOARCHIVELOG mode, the online redo log files are overwritten each time a log switch occurs. LGWR will not overwrite a redo log group until the checkpoint for that group is complete. This ensures that committed data can be recovered if there is an instance crash. During the instance crash, only the SGA is lost. There is no loss of disks, only memory. For example, an operating system crash causes an instance crash.

Archiving Redo Log Files (continued)

ARCHIVELOG Mode If the database is configured to run in ARCHIVELOG mode, inactive groups of filled online redo log files must be archived before they can be used again. Because changes made to the database are recorded in the online redo log files, the DBA can use the physical backup of the data files and the archived online redo log files to recover the database without losing any committed data because of any single point of failure, including the loss of a disk. Usually a production database is configured to run in ARCHIVELOG mode.

ARC0 Process

The ARC0 process initiates backing up, or archiving, of the filled log group at every log switch. It automatically archives the online redo log before the log can be reused, so that all of the changes made to the database are preserved. This enables the DBA to recover the database to the point of failure, even if a disk drive is damaged.

Summary

Summary

In this lesson, you should have learned how to:

- Explain database files: data files, control files, online redo logs
- Explain SGA memory structures: DB buffer cache, shared SQL pool, and redo log buffer
- Explain primary background processes: DBW0, LGWR, CKPT, PMON, SMON, and ARC0
- Explain SQL processing steps: parse, execute, fetch

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE

Oracle Files

The Oracle database includes these files:

- Control files: Contain information required to verify the integrity of the database, including the names of the other files in the database (The control files are usually mirrored.)
- Data files: Contain the data in the database, including tables, indexes, rollback segments, and temporary segments
- Online redo logs: Contain the changes made to the data files (Online redo logs are used for recovery and are usually mirrored.)

Other files commonly used with the database include:

- Parameter file: Defines the characteristics of an Oracle instance
- Password file: Authenticates privileged database users
- Archived redo logs: Are backups of the online redo logs

SGA Memory Structures

The System Global Area (SGA) has three primary structures:

- Shared pool: Stores the most recently executed SQL statements and the most recently used data from the data dictionary
- Database buffer cache: Stores the most recently used data
- Redo log buffer: Records changes made to the database using the instance

Background Processes

A production Oracle instance includes these processes:

- Database Writer (DBW0): Writes changed data to the data files
- Log Writer (LGWR): Records changes to the data files in the online redo log files
- System Monitor (SMON): Checks for consistency and initiates recovery of the database when the database is opened
- Process Monitor (PMON): Cleans up the resources if one of the processes fails
- Checkpoint Process (CKPT): Updates the database status information after a checkpoint
- Archiver (ARC0): Backs up the online redo log to ensure recovery after a media failure (This process is optional, but is usually included in a production instance.)

Depending on its configuration, the instance may also include other processes.

SQL Statement Processing Steps

The steps used to process a SQL statement include:

- Parse: Compiles the SQL statement
- Execute: Identifies selected rows or applies DML changes to the data
- Fetch: Returns the rows queried by a SELECT statement

2

Getting Started with the Oracle Server

Objectives

Objectives

After completing this lesson, you should be able to do the following:

- **Identify the features of the Universal Installer**
- **Set up operating system and password file authentication**
- **List the main components of Oracle Enterprise Manager and their uses**

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

Overview

Database Administration Tools: Examples	
Tool	Description
SQL*Plus Line Mode	A utility used for administrative tasks such as starting up, shutting down, or recovering a database
Oracle Enterprise Manager	Graphical user interface to administer, monitor, and tune one or more databases
SQL*Loader	Utility for loading data from external files into Oracle tables
Export or Import utility	Utility for exporting or importing data in Oracle format
Password File utility	Utility for creating database password file

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE

Examples of Common Database Administration Tools

The table lists examples of the common database administration tools, and tasks that a database administrator (DBA) can perform using the tools.

These tools will be covered in this course, but they are only a subset of the utilities supplied by Oracle. The commands for starting up the different tools depend on the platform.

This lesson introduces the use of the following tools that assist a DBA in performing administrative tasks:

- Universal Installer
- Oracle Enterprise Manager
- Oracle Configuration Assistant

Note: In addition to these tools, utilities such as the Database Configuration Assistant for creation of an Oracle database will be covered in the lessons “Managing an Oracle Instance” and “Creating a Database.”

Oracle Universal Installer

Oracle Universal Installer: Features

- **Based on a Java engine**
- **Automatic dependency resolution and complex logic handling**
- **Installation from the Web**
- **Component and suite installations**
- **Implicit deinstallation**
- **Support for multiple Oracle homes**
- **NLS/globalization support**

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE[®]

Features of the Oracle Universal Installer

The Java-based Oracle Universal Installer offers an installation solution for all Java-enabled platforms, allowing for a common installation flow and user experience independent of the platform.

The new installation engine automatically detects dependencies among components and performs an installation accordingly, depending on the products and the type of selected installation. Bundle and suite installations, a predefined set of products and their order, are easily identified with a minimum number of user dialogs and no duplicate information requested.

The Universal Installer can be used to point to a URL where a release or staging area was defined and install software remotely over HTTP.

The deinstallation products installed using the Universal Installer are built into the engine itself. The deinstallation actions are the “undo” of installation actions.

The Universal Installer maintains an inventory of all the Oracle homes on a target machine, their names, products, and version of products installed on them.

The Universal Installer detects the language of the operating system and runs the installation session in that language.

Silent Installation Using Response Files

- To start Universal Installer on Windows NT:

```
setup.exe -responsefile filename [-silent]  
[-nowelcome]
```

- To start Universal Installer on Solaris:

```
runInstaller -responsefile filename  
[-silent] [-nowelcome]
```

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Silent Installation Using Response Files

Response files are collections of variable settings that provide values that would have been asked of the user otherwise. For a particular component installation, the Universal Installer can read these values from a predefined response file. With this file as an additional argument, the Universal Installer can create a smaller number of dialog installation sessions or a complete noninteractive installation session.

- For Windows NT: The installation program is called `setup.exe` and is located in the `Program Files\Oracle\oui\install` directory.
- For UNIX: The installation program is called `runInstaller` and is located in the `INSTALL\install\solaris` directory.

For example, to run the installer for Windows NT from a CD-ROM located on the D drive:

```
D:\> setup
```

Optimal Flexible Architecture (OFA)

- **Organizes software and data on disk**
- **Facilitates routine administrative tasks**
- **Alleviates switching among multiple Oracle databases**
- **Adequately manages and administers database growth**
- **Minimizes resource contention**
- **Is similar on NT and UNIX**

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Optimal Flexible Architecture (OFA)

Installation and configuration on all supported platforms now complies with Optimal Flexible Architecture (OFA). OFA organizes database files by type and usage. Binary files containing Oracle base code are installed in one directory; control files, log files, and administrative files are installed in a separate directory; and database files are installed in a separate directory.

OFA provides:

- A structured method for installing Oracle databases and applications
- Ease of administration, such as for backup and recovery, through a better file layout structure
- Better performance by decreasing disk contention for data files, binary files, and administrative files, which can now reside on separate directories or disks
- Easier administration of multiple Oracle homes on the same machine by separating files on different disk devices and directories

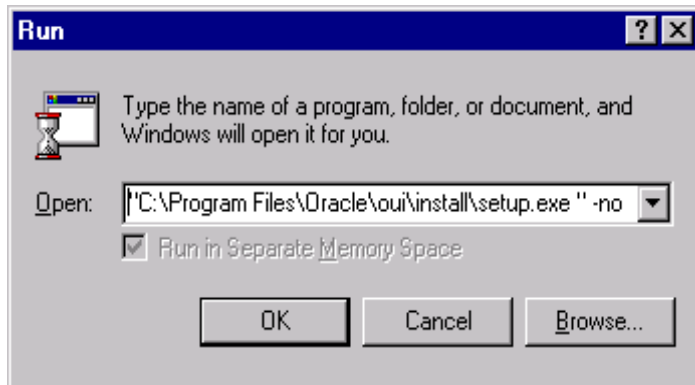
How to Launch the Universal Installer

Try It 2-1 Launch the Universal Installer and list your installed products.

- 1 Launch the Universal Installer:
Start—>Programs—>Oracle Installation Products—>Universal Installer
- 2 Click Installed Products in the Welcome window to list your installed products.
- 3 Expand an Oracle Home and check the installed products and version. When you have finished, click Close.
- 4 Click Exit to exit the Universal Installer. Click Yes in the dialog box.

Try It 2-2 Launch the Universal Installer, but skip the welcome page.

- 1 Launch the Universal Installer with the `-nowelcome` option:
Start—>Run.
- 2 Enter `"C:\Program Files\Oracle\oui\install\setup.exe" -nowelcome`, and click OK.



- 3 Click Exit to exit the Universal Installer. Click Yes in the dialog box.

Validating Privileged Users

Database Administrator Users

The two default database administrator users `SYS` and `SYSTEM` are:

- Created automatically
- Granted the DBA role

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE®

The Database Administrator Users

Extra privileges are necessary to execute administrative duties on the Oracle server, such as creating users. These operations must be performed by database administrators.

Two database user accounts, `SYS` and `SYSTEM`, are created automatically with the database and granted the DBA role—that is, a predefined role that is created automatically with every database. The DBA role has all database system privileges.

Note: This subject is covered in more detail in the lesson “Managing Privileges.”

Users `SYS` and `SYSTEM`

`SYS`

- **Password:**
`change_on_install`
- **Owner of the database data dictionary**

`SYSTEM`

- **Password:**
`manager`
- **Owner of additional internal tables and views used by Oracle tools**

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

`SYS`

When a database is created, the user `SYS`, identified initially by the password `change_on_install`, is created automatically and granted the DBA role.

All of the base tables and views for the data dictionary are stored in the schema `SYS`.

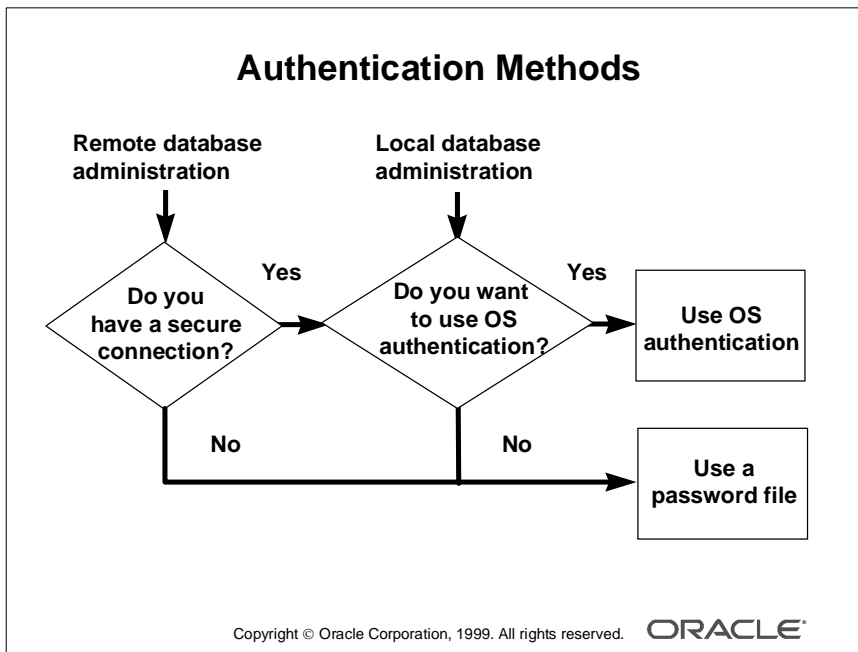
`SYSTEM`

When a database is created, the user `SYSTEM`, identified initially by the password `manager`, is also created automatically and granted the DBA role privileges for the database.

Additional tables and views owned by the user `SYSTEM` are created. They contain administrative information used by Oracle tools.

Both `SYS` and `SYSTEM` are regular users, which are stored in the data dictionary; they can only log on when the database is open.

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



Connecting with Administrator Privileges

In some cases, the database administrator needs a special authentication method because the database may not be open, especially for operations such as shutdown and startup.

Depending on whether you want to administer your database locally on the same machine on which the database resides or to administer many different database servers from a single remote client, you can choose either operating system authentication or password files to authenticate database administrators.

Operating System Authentication

- Set up the user to be authenticated by the operating system.
- Set `REMOTE_LOGIN_PASSWORDFILE` to `NONE`.
- Use the following commands to connect to a database:

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

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE®

How to Use Operating System Authentication

Setting up the user to be authenticated by the UNIX operating system is different from setting up a user for authentication on a Windows NT system.

UNIX

- 1 The user must be a member of the UNIX group, usually called *dba*, that is created before the Oracle server is installed. The installer assigns Oracle database administrator and operator privileges to this UNIX group.
- 2 View the `/etc/group` and `/etc/passwd` files to determine the members of the UNIX group.

The following lines are an excerpt from the `/etc/passwd` file:

```
user15:x:1064:100::/home/disk3/user15:/bin/ksh
oracle:x:920:100::/export/home/oracle:/bin/ksh
vvijayan:x:1032:100::/users/vvijayan:/bin/ksh
```

The following line belongs to the `/etc/group` file:

```
dba::100:root,oracle,estrodac,tigger,jdlianni
```

- 3 Make sure that the value of the `REMOTE_LOGIN_PASSWORDFILE` parameter is `NONE`. The default value for this parameter is `EXCLUSIVE` with version 8.1.x or later. In earlier versions of Oracle, the default value was `NONE`.

UNIX (continued)

- 4 Connect to the database with the privilege SYSDBA or SYSOPER. These are special database administrator privileges. The slash implies connecting in the schema owned by SYS:

```
CONNECT / AS {SYSDBA |SYSOPER}
```

Note: Using the privileges SYSDBA and SYSOPER is covered in the lesson “Managing Privileges.”

Windows NT

- 1 Create a new local Windows NT users’ group called ORA_<SID>_DBA and ORA_<SID>_OPER that is specific to an instance, or ORA_DBA and ORA_OPER that is not specific to an instance.
- 2 Add a Windows NT operating system user to that group. Once you access this domain, you are automatically validated as an authorized DBA.
- 3 Ensure that you have the following line in your `sqlnet.ora` file:
SQLNET.AUTHENTICATION_SERVICES = (NTS)
- 4 Set the REMOTE_LOGIN_PASSWORDFILE parameter to NONE.
- 5 Connect to the database with the privilege SYSDBA or SYSOPER:

```
CONNECT / AS { SYSDBA|SYSOPER }
```

Note

- To connect to a Windows NT server from a local, remote Windows NT or Windows 95 client, Net8 should be installed on both the client and the server, but a JDBC connection can also be used.
- The CONNECT INTERNAL command, used with earlier versions of Oracle, has been replaced by the syntax:

```
CONNECT INTERNAL/pw AS SYSDBA
```

CONNECT INTERNAL continues to be supported for backward compatibility only.

Using Password File Authentication

- Create the password file using the password utility:

```
$orapwd file=$ORACLE_HOME/dbs/orapwSID
password=admin entries=10
```

- Set **REMOTE_LOGIN_PASSWORDFILE** to **EXCLUSIVE** or **SHARED**.
- Use the following command to connect to a database:

```
CONNECT INTERNAL/ORACLE
```

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

How to Use Password File Authentication

Oracle provides a password utility that allows connection to the Oracle server using a standard username and password, but the utility connects the user to the SYS schema rather than to the username provided. Access to the database using the password file is provided by special GRANT commands issued by privileged users (see the lesson “Managing Privileges”).

Using a Password File on a UNIX and a Windows NT Server

- 1 Create the password file using the password utility ORAPWD.

```
orapwd file=fname password=password entries=entries
```

where:

fname	is the name of the password file
password	is the password for SYS and INTERNAL
entries	is the maximum number of distinct database administrators

The following command creates a password file with the password `admin` for the user `SYS` and `INTERNAL` and accepts up to five users with different passwords:

```
On UNIX: $orapwd file=$ORACLE_HOME/dbs/orapwU15 password=admin
entries=10
```

How to Use Password File Authentication (continued)

Using a Password File on a UNIX and a Windows NT Server (continued)

On Windows NT: `C:\>orapwd file=%ORACLE_HOME%/database/pwdSID
password=admin entries=10`

- 2 Set the `REMOTE_LOGIN_PASSWORDFILE` parameter to `EXCLUSIVE` or `SHARED`.

where:

<code>EXCLUSIVE</code>	indicates that only one instance can use the password file and that the password file contains names other than <code>SYS</code> and <code>INTERNAL</code>
<code>SHARED</code>	indicates that more than one instance can use the password file (The only users recognized by the password file are <code>SYS</code> and <code>INTERNAL</code> .)

- 3 Connect to the database as follows:

```
SQL> CONNECT internal/oracle
```

Note

- On UNIX
The password files are usually located in the `$ORACLE_HOME/dbs` directory on UNIX.
- On Windows NT
 - The password file is usually located in the `%ORACLE_HOME%\DATABASE` directory. You can specify a nondefault location of the password file in the NT registry with the key `ORA_SID_PWFIL`.
 - The password for `INTERNAL` is `Oracle`, if Oracle is installed through the Oracle8i Enterprise Edition option. You can set the password during installation by using the Custom installation option.

Maintaining the Password File

On UNIX and Windows NT, delete the existing password file and create a new password file by using the `ORAPWD` utility.

Oracle Enterprise Manager

What Is Oracle Enterprise Manager?

- **The Oracle unified system management enterprise framework**
- **Central view of complete managed system**
- **Set of underlying services**
- **Free-of-charge DBA Management Pack**
- **Performance, tuning, diagnostics, and change management packs**
- **Application Management Pack**

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

What Is Oracle Enterprise Manager?

The Oracle Enterprise Manager is a unified management enterprise framework for the Oracle environment.

The Oracle Enterprise Manager console provides a global view of the system. It includes both hierarchical tree and graphical representations of the objects and their relationships in the system.

Intelligent Agent processes residing on managed nodes facilitate remote management of jobs, events, and status. These agents make autonomous “lights out” management possible.

The common services (job, events, discovery, and security services) complete the Oracle Enterprise Manager framework, providing the base functionality behind all Oracle Enterprise Manager applications.

Oracle Enterprise Manager also includes sets of specialized management applications.

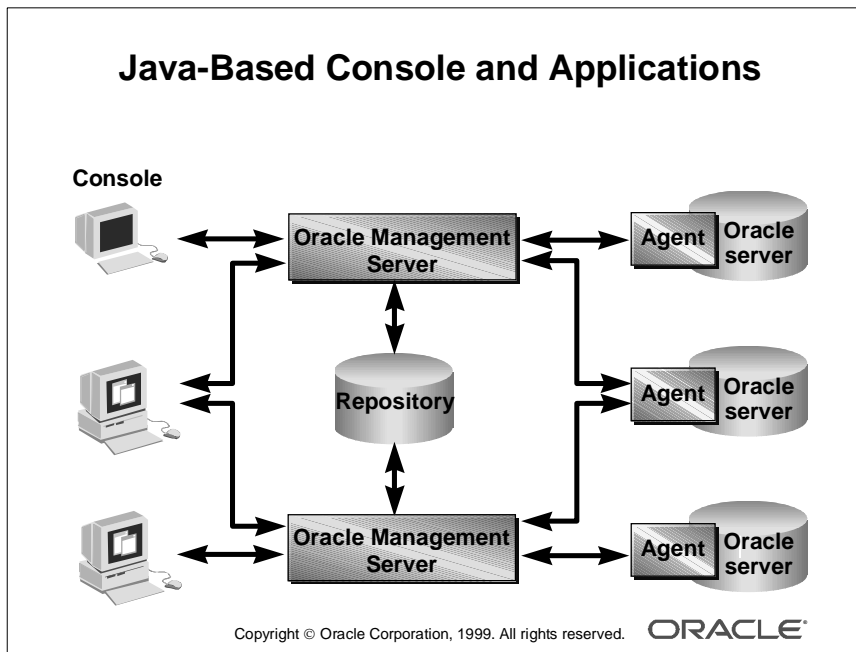
- **DBA Management Pack:** Comes with the database for free
- **Advanced Management Packs:** Tuning Pack, Diagnostics Pack, and Change Management Pack
- **Application Management Pack:** Support for Oracle Applications Concurrent Manager, WebForm Servers, and Workflow subsystems

What Is Oracle Enterprise Manager? (continued)

The framework is extensible, providing a rich set of APIs that allow for integration of any management application, including management for the complete set of Oracle products or any partner or customer products.

Technical Note

The Application Management Pack was not released with Oracle Enterprise Manager 2.0.4. It will be released later.



Oracle Enterprise Manager Architecture

Version 2 of Oracle Enterprise Manager extends the client-server architecture introduced with version 1 to a highly scalable three-tier model.

The first tier consists of a Java-based console and integrated applications that can be installed or run from a Web browser.

Oracle Management Server

The second-tier component of Oracle Enterprise Manager version 2 is the Oracle Management Server (OMS). The main function of the OMS is to provide centralized intelligence and distributed control between clients and managed nodes, processing and administering all system tasks. As the number of managed systems increases, the architecture scales through the addition of OMSs. Failover and load balancing is also automated within the OMS, providing much greater reliability in notification processing.

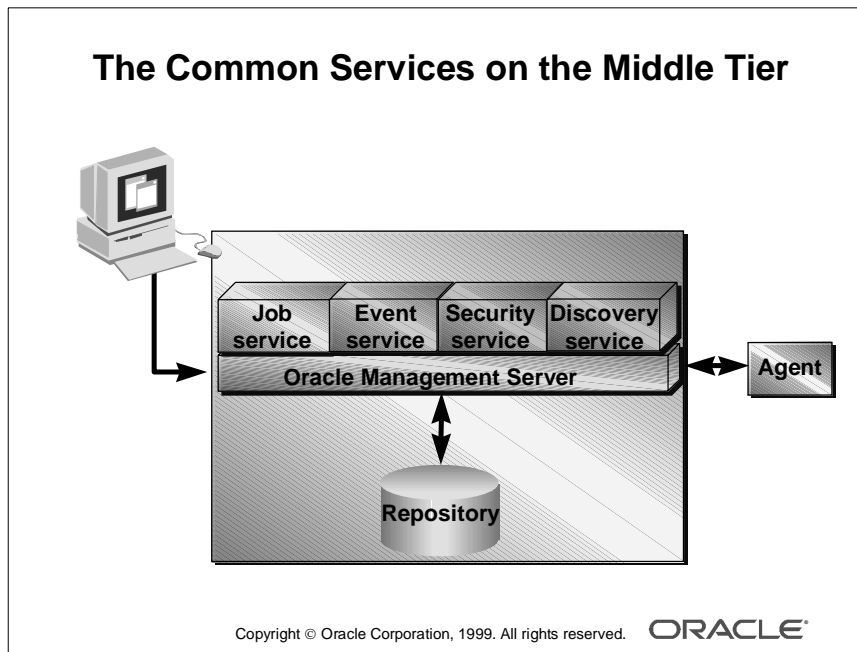
Oracle Enterprise Manager Repository

The OMS uses the Oracle Enterprise Manager repository as its persistent back-end store. This repository maintains system data, application data, and the state of managed entities distributed throughout the environment. Version 2 allows multiple users to access and share repository data for systems where responsibilities are shared.

Oracle Enterprise Manager Repository (continued)

The third tier is composed of targets, such as databases, nodes, or other managed services. The Intelligent Agent functions as the executor of jobs and events sent by the OMS.

Client-server connections to the database by way of the Oracle Enterprise Manager console are still supported.



Common Services

Oracle Enterprise Manager is made up of services that help you manage nodes throughout your network.

Discovery Service Oracle Enterprise Manager automatically discovers (locates) all the databases and other services running on the managed nodes, once the nodes are identified. These services includes Web servers, listeners, machines, parallel servers, video servers, and other services, in addition to databases.

Job Scheduling System The job scheduling system provides job storing and forwarding capability that enables the automation of standard and repetitive tasks. You can create and manage jobs, schedule their execution, and view and share information about defined jobs with other administrators. The system can also notify you and other administrators through electronic mail or paging upon job completion or failure.

Common Services (continued)

Event Management System You can use the event management system to monitor your network environment for operational occurrences such as loss of service, borderline conditions such as shortage of storage, and capacity problems such as high CPU usage.

When an event is detected, you or a specified administrator can be notified, or a “fix-it” job can run in response to an event.

Security Security parameters are defined for services, objects, and administrators. For example, a super administrator can create new administrators and define the administrator privileges.

Shared Repository

- Multiple administrators share information in one central repository
- Administrator access controls:
 - Superusers
 - Regular users
- One default superuser: `sysman/oem_temp`
- Administrators are maintained from the console

Copyright © Oracle Corporation, 1999. All rights reserved.

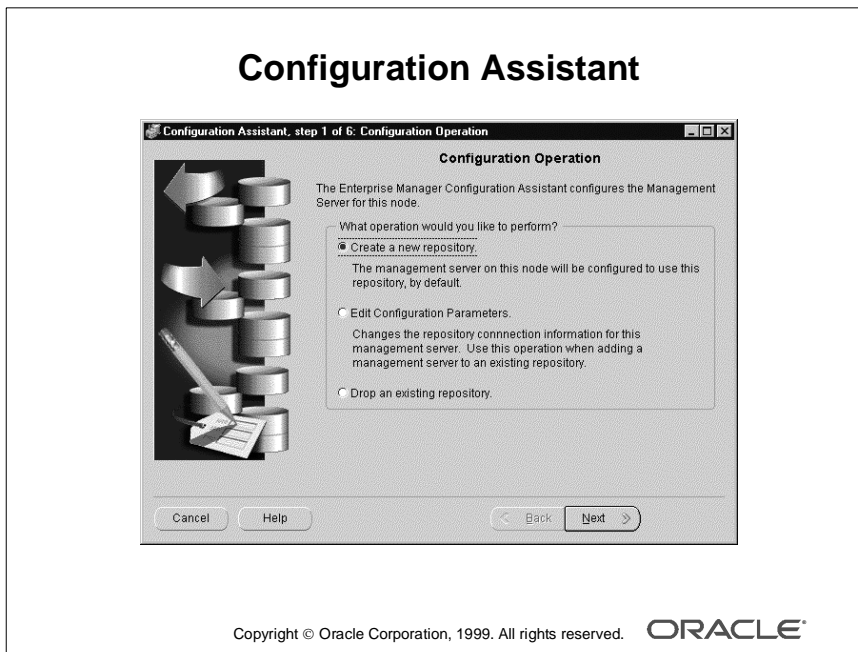
ORACLE®

Multuser System

Oracle Enterprise Manager version 2 introduces a multuser system. Each administrator has a separate account with which to log in to the console. Depending on the permission settings, the administrator has access to the data stored in the central repository, a repository shared by all the Oracle Enterprise Manager administrators for the managed environment.

Superusers have advanced permissions and define permission settings for other administrators.

Note: When Oracle Enterprise Manager is installed, the default superuser `sysman/oem_temp` is created. A superuser has full privileges and can be used to create new superusers or regular users.



Starting the Enterprise Manager Configuration Assistant

The Configuration Assistant tool creates shared repositories, configures the OMS, and sets up the local console. It is started automatically from the Universal Installer after the installation has finished, and it can also be started manually from:

- The operating system prompt at any stage by running:
`%emrepmgr`
- The Windows NT Start menu by using Start—>Programs—>Oracle - *EM Home*—>Oracle Enterprise Management—>Enterprise Manager Configuration Assistant.

Technical Note

Make sure that all databases that will contain a shared repository have the necessary tablespaces created before using the Configuration Assistant. It is recommended that each repository should be in a separate tablespace for ease of management.

How to Create a New Repository

Try It 2-3 The Configuration Assistant is the tool used to create or drop an Oracle Enterprise Manager repository. It can also be used to modify the repository connection for an Oracle Management Server (OMS).

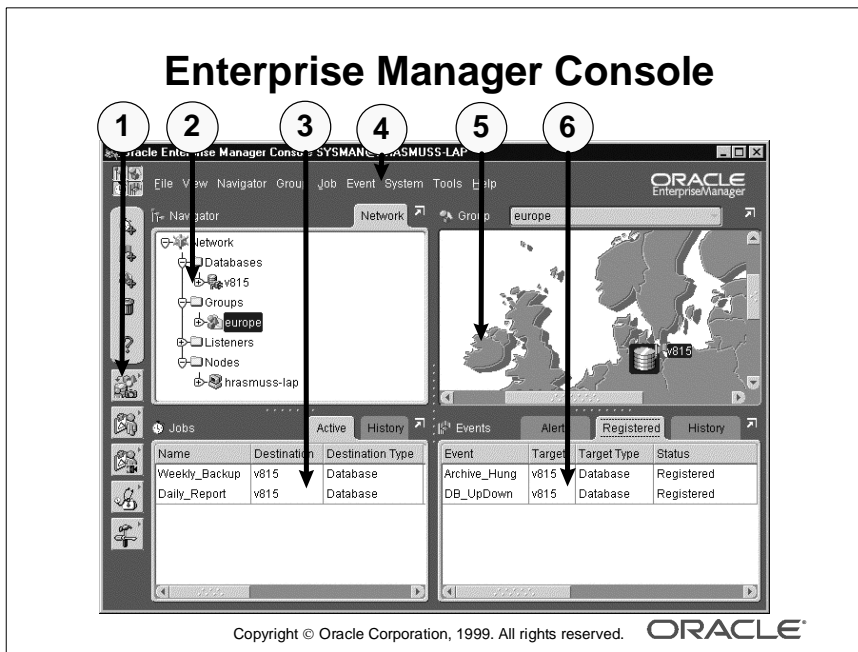
Create a repository using the Configuration Assistant.

- 1 Launch the Configuration Assistant:
Start—>Programs—>Oracle - *EMV2 Home*—>Oracle Enterprise Management
—>Enterprise Manager Configuration Assistant
- 2 Ensure that the “Create a new repository” option button is selected, and click the Next button.
- 3 Enter the username `system`, the password `manager`, and the service `host:port:sid (JDBC)`, and click Next.
- 4 If you get a warning that your processes should be 200, click Yes to continue.
- 5 Enter the username `Rep_Usernumber` and the password `manager`, confirm the password `manager`, and click Next.
- 6 Specify the default tablespace `OEM` and the temporary tablespace `TEMP`, and click Next.
- 7 Click Finish and wait for the repository to be built. Click Close to exit the Configuration Assistant.

Try It 2-4 The Oracle Management Server (OMS) must be started after creating the repository. It should start automatically on Windows NT when you start up the PC.

Start the OMS.

- 1 Start the OMS service:
Start—>Settings—>Control Panel
- 2 Double-click Services and select the Oracle*EMV2 Home*Management Server service. If it is not started, click Start to start it.



The Enterprise Manager Console

The Console provides a global view of the system. It includes both a hierarchical tree and a graphical representation of the objects in the system. It has the following features:

- 1 A *set of drawers* invokes other applications to perform administrative tasks. It provides an alternative to using the menu.
- 2 A *navigator* or object explorer view provides a hierarchical view of Oracle services on the network. The navigator permits administrators to browse the different Oracle services, such as databases, listeners, nodes, and name servers, and to modify the characteristics of objects; for example, users and the tables that they contain.
- 3 A *job system* permits remote execution of tasks related to listeners, databases, or the host itself. The job system is based on the procedural Tool Control Language (TCL) engine.
- 4 A *menu* can be used to initiate other administrative applications and perform various tasks.
- 5 A *map* or topographical view permits Oracle services to be grouped based on spatial relationships, function, or both. The map view enables users to focus on managed objects directly within their purview or interest.
- 6 An *event system* monitors and reports on system status. The event system in conjunction with the job system will take corrective action based on predefined criteria and can advise the administrator by pager or electronic mail that a particular event has occurred.

How to Launch the Oracle Enterprise Manager Console

Try It 2-5 Launch the console to get a global view of the system belonging to the connected repository.

- 1 Launch the console:
Start—>Programs—>Oracle - *EMV2 Home*—>Oracle Enterprise Management
—>Enterprise Manager Console
- 2 Enter the administrator `sysman`, the password `oem_temp`, and the management server *your PC hostname*, and click OK.
- 3 When asked to change the password, enter `manager` as your new password.

Try It 2-6 To view a service in the navigator tree, it must be discovered by the agent on the node where the service resides.

- 1 Select Navigator—>Discover Nodes... from the Console menu.
- 2 Enter *your Server hostname*, and click OK. Click Close when discovered. Your database should now be in your navigator tree.
- 3 Expand the Database folder to see if your database has been discovered in the navigator tree.

DBA Management Pack

- **Instance Manager**
- **Security Manager**
- **Storage Manager**
- **Schema Manager**
- **SQL*Plus Worksheet**
- **Tools and wizards for data management and backup**

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

The DBA Management Pack

The standard applications that are supplied with Oracle Enterprise Manager include the following:

- **Instance Manager:** Used to control database availability and define initialization parameters to specify the characteristics of the instance
- **Security Manager:** Used to manage users and privileges
- **Storage Manager:** Used to organize the database files and manage rollback segments
- **Schema Manager:** Used to create and maintain objects such as tables, indexes, and views
- **SQL*Plus Worksheet:** A command line interface that can be used to run SQL commands, PL/SQL code, SQL*Plus commands, and special DBA commands, such as `startup` and `shutdown`
- **Tools and wizards for data management:** Used to load and reorganize data in databases
- **Tools and wizards for backup management:** Used to back up, restore, and recover databases, and to manage redo log files

The DBA Management Pack (continued)

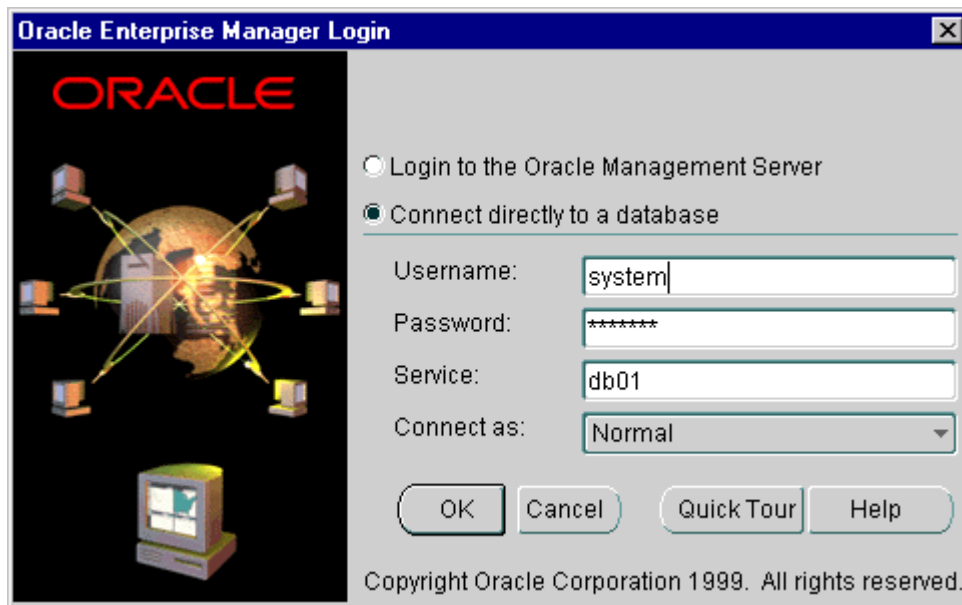
Note

- All applications can be invoked either from the console or directly from the operating system.
- The use of these applications to perform specific tasks is covered in the other lessons throughout the course.
- The use of backup management tools and wizards to back up, restore, and recover databases is discussed in the course *Enterprise DBA Part 1B: Backup and Recovery*.

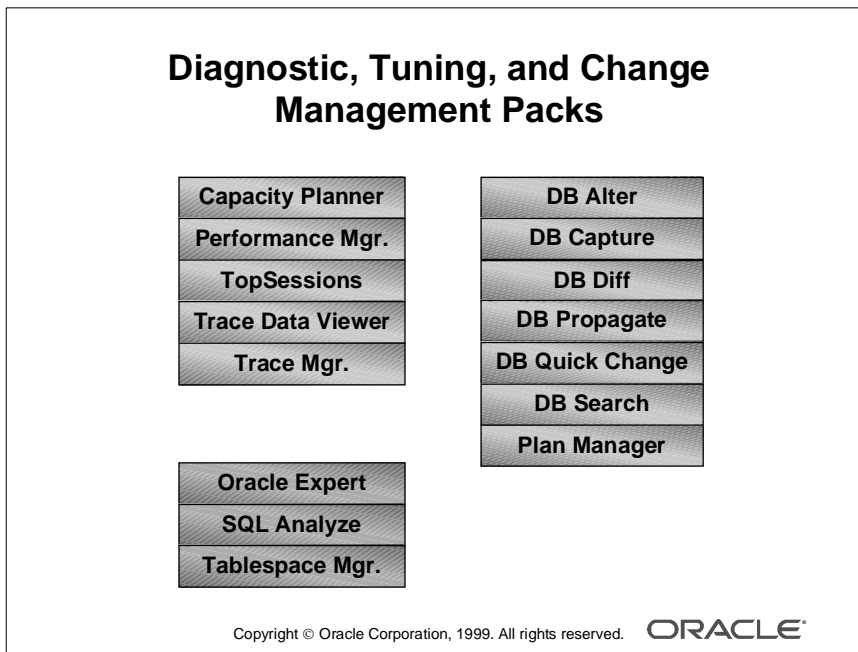
How to Invoke the SQL*Plus Worksheet

1 Launch the SQL*Plus Worksheet:

Start—>Programs—>Oracle - EMV2 Home—>DBA Management Pack
—>SQL Plus Worksheet



- 2 Make sure you connect directly to a database. Enter the username `system`, the password `manager`, and the service name for your working database, and click OK.
- 3 Select File—>Exit to close the worksheet.



Management Packs

There are three management packs, which contain tools to assist in diagnosing, tuning, and managing change in an Oracle database environment.

- The Diagnostics Pack contains the following tools:
 - Oracle Trace and Data Viewer: Used to create, schedule, administer, and view trace information collected from the database, Net8, and other applications
 - Performance Manager: Used to create, view, and edit graphical charts to monitor and display database information
 - Capacity Planner: Collects database and operating system performance statistics, stores historical database information, and analyzes data to plan future capacity requirements
 - TopSessions: Views details about sessions currently connected to a database
 - Advanced Events: Additional Oracle Enterprise Manager events to assist with proactive database administration

Management Packs (continued)

- The Tuning Pack contains the following tools:
 - Oracle Expert: Optimizes Oracle database performance by collecting parameter, environment, application, and database structure information (The collected data is analyzed and tuning recommendations are made.)
 - Tablespace Manager: Detects and fixes space management problems, and reorganizes database objects
 - SQL Analyze: Assists in locating, analyzing, and editing SQL statements to improve application performance
- The Change Management Pack contains the following tools:
 - Database Capture: Captures sets of database object definitions (baselines)
 - Database Diff: Compares differences in object definitions between two databases (or baselines)
 - Database Quick Change: Used to make changes to one object definition in one database
 - Database Alter: Used to make changes to one or more object definitions in one or more databases
 - Database Propagate: Used to select one or more object definitions in a database and then propagate those definitions in another schema or database
 - Plan Manager: Used to centralize all the change management applications

Note: The terms *tablespace*, *segment*, *extent*, and *free space* will be discussed in subsequent lessons.

Summary

Summary

In this lesson, you should have learned how to:

- **Describe features of the Universal Installer**
 - Based on a Java engine
 - Supports multiple Oracle homes
 - Runs in silent mode
- **Explain setup of operating system and password file authentication**
- **Describe main components of Oracle Enterprise Manager**
 - Three-tier architecture
 - Java-based console and applications
 - Multiuser system with shared repositories
 - Single point of control

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

3

Managing an Oracle Instance

Objectives

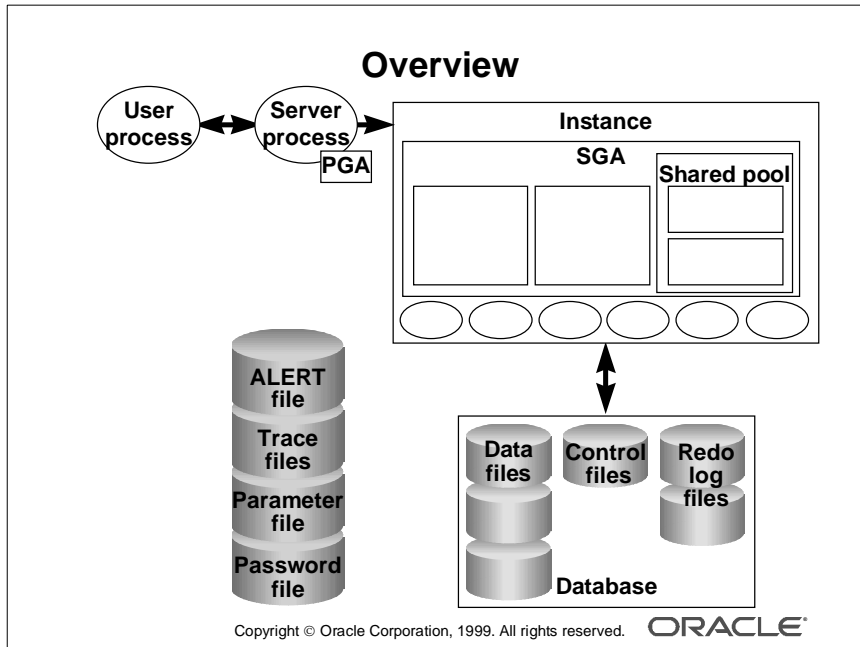
Objectives

After completing this lesson, you should be able to do the following:

- **Create the parameter file**
- **Start up an instance and open the database**
- **Close a database and shut down the instance**
- **Get and set parameter values**
- **Manage sessions**
- **Monitor the ALERT file and the trace files**

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

Overview



Overview of Starting Up and Shutting Down an Oracle Server

An Oracle database is not available to users until the database administrator has started the instance and opened the database.

During a database startup, the following events occur. Each event takes the Oracle database through various stages:

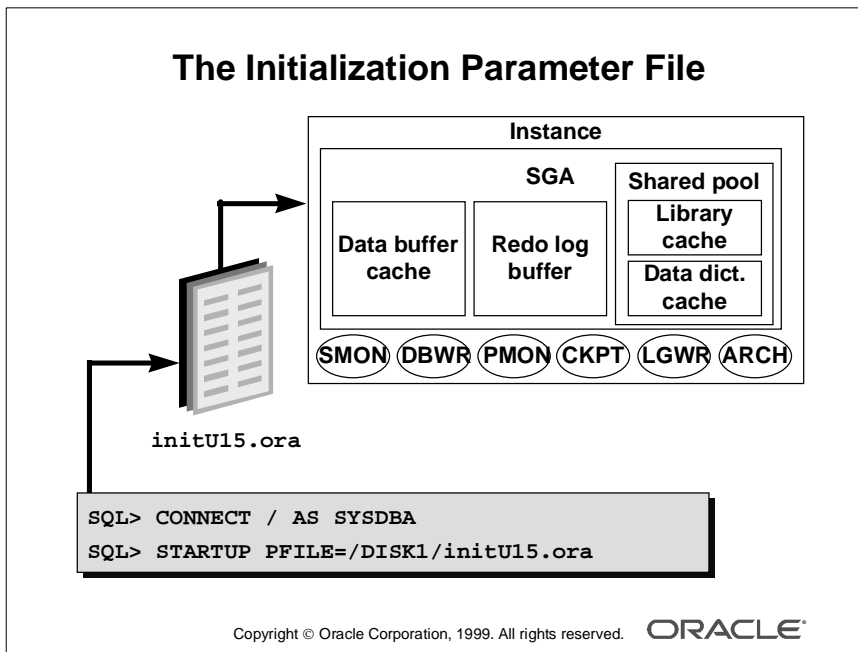
- 1 Start an instance.
- 2 Mount the database.
- 3 Open the database.

Every time an instance is started, Oracle uses a parameter file, which contains initialization parameters, to allocate the System Global Area (SGA) and to start the background processes.

If an instance is started or a database is open, you can follow these steps to shut down the database:

- 1 Close the database.
- 2 Dismount the database.
- 3 Shut down the instance.

When a database is closed, users cannot access it.



Creating and Using the Parameter File

The parameter file, commonly referred to as the `init \textit{sid} .ora` file, is a text file that can be maintained using a standard operating system editor.

By default, it is located in the `$ORACLE_HOME/dbs` directory on a UNIX machine and in the `%ORACLE_HOME%\database` directory on Windows NT. With Oracle8i on Windows NT, the parameter file points to the `%ORACLE_HOME%\admin\sid\pfile` directory where the actual parameter file is stored. This is done by using the `IFILE` parameter.

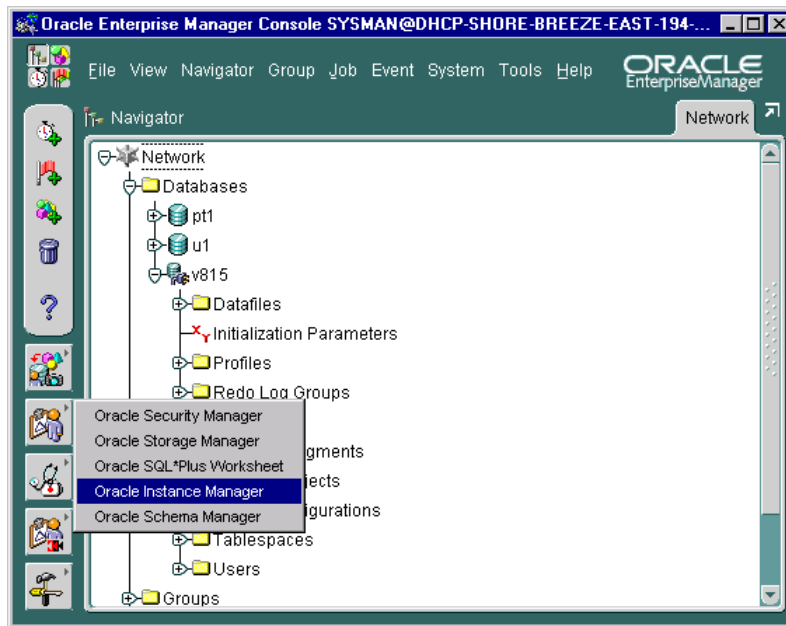
The parameter file is read only during instance startup. If the file is modified, shut down and restart the instance to make the new parameter values effective. Some parameters are dynamic, which means that they can be modified while the instance is running. Dynamic parameters are covered later in this lesson.

The Oracle Enterprise Manager Console or Instance Manager allows the DBA to change and view the initialization parameters. They can be stored either in a local parameter file or in the Oracle Enterprise Manager repository by using stored configurations. If using stored configurations, the DBA must be connected by the way of an Oracle Management Server (OMS) to get access to a repository. In earlier versions of Oracle Enterprise Manager (1.x) the initialization parameters were stored locally in the Windows NT registry.

How to Create a Stored Configuration

Try It 3-1 Launch Instance Manager and save a stored configuration.

- 1 Launch the Oracle Enterprise Manager Console:
Start—>Programs—>Oracle - *EMV2 Home*—>Oracle Enterprise Management
—>Enterprise Manager Console
- 2 Enter the administrator *sysman*, the password *manager*, and the management server *your PC hostname*, and click OK.
- 3 Click the second drawer on the left side of the console and select Instance Manager.



- 4 Expand your working database and select the initialization parameters in the navigator tree.
- 5 Your initialization parameters should now be displayed in the right pane of the window. Click Save and name your configuration before clicking OK to save a stored configuration.
- 6 Expand the stored configuration folder to verify that your parameters were saved.

Uses of Parameters

- **Size the System Global Area (SGA).**
- **Set database and instance defaults.**
- **Set user or process limits.**
- **Set limits on database resources.**
- **Define various physical attributes of the database, such as the database block size.**
- **Specify control files, archived log files, the ALERT file, and trace file locations.**

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Uses of Parameters

The parameters in the `initsid.ora` file can have a significant effect on database performance, and some need to be modified in the following ways for production systems:

- Size the System Global Area (SGA) components to optimize performance.
- Set database and instance defaults.
- Set user or process limits.
- Set limits on database resources.
- Define (on database creation only) various physical attributes of the database, such as the database block size.
- Specify control files, archived log files, ALERT file, and trace file locations.

Parameter File Example

```
# Initialization Parameter File: initU15.ora
db_name                = U15
control_files          = (/DISK1/control01.con,
                          /DISK2/control02.con)

db_block_size          = 8192
db_block_buffers       = 2048
shared_pool_size       = 52428800
log_buffer             = 64K
processes              = 50
db_files               = 1024
log_files              = 10
max_dump_file_size     = 10240
background_dump_dest   = (/home/disk3/user15/BDUMP)
user_dump_dest         = (/home/disk3/user15/UDUMP)
core_dump_dest         = (/home/disk3/user15/CDUMP)
rollback_segments      = (r01,r02,r03,r04,r05,r06,r07,r08)
...
```

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Rules for Specifying Parameters

- Specify the values in the following format: keyword=value.
- All parameters are optional.
- The server has a default value for each parameter. This value may be operating system dependent, depending on the parameter.
- Parameters can be specified in any order.
- Comment lines begin with the # symbol.
- Enclose parameters in double quotation marks to include character literals.
- Additional files can be included with the keyword IFILE.
- If case is significant for the operating system, then it is also significant in filenames.
- Multiple values are enclosed in parentheses and separated by commas.

Note: Develop a standard for listing parameters; either list them alphabetically or group them by functionality.

Parameters That Should Be Specified

Parameter	Description
BACKGROUND_DUMP_DEST	Location where background process trace files are written (LGWR, DBWn, and so on). This is also the location for the alert log.
COMPATIBLE	Version of the server with which this instance should be compatible. The default is 8.1.0.
CONTROL_FILES	Names of the control files.
DB_BLOCK_BUFFERS	Number of blocks cached in the SGA.
DB_NAME	Database identifier of eight characters or fewer. This is the only parameter that is required when creating a new database.
SHARED_POOL_SIZE	Size in bytes of the shared pool.
USER_DUMP_DEST	Location where user debugging trace files are created on behalf of a user process.

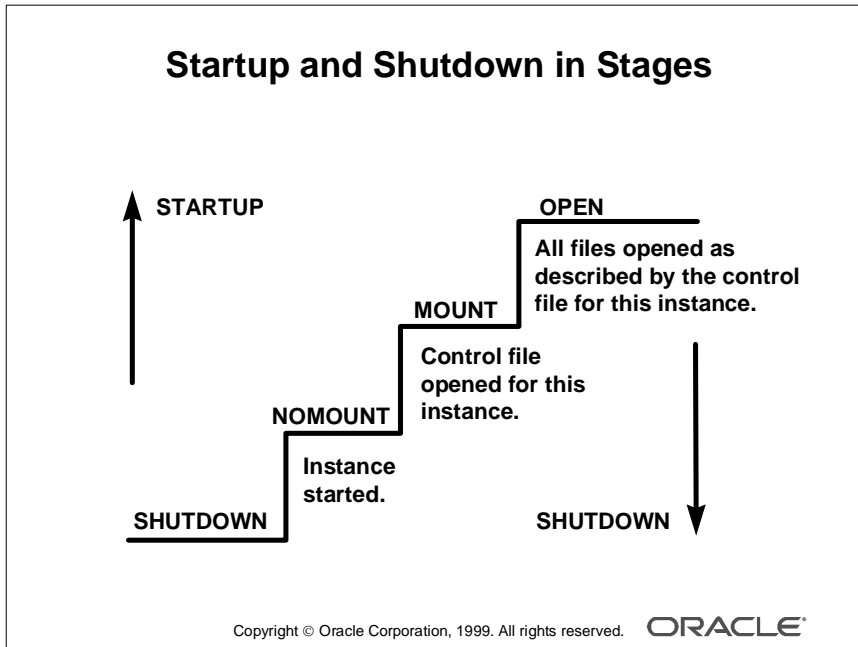
Technical Note

The default values depend on the version of the Oracle server.

Commonly Modified Parameters

Parameter	Description
IFILE	Name of another parameter file to be embedded within the current parameter file. Up to three levels of nesting is possible.
LOG_BUFFER	Number of bytes allocated to the redo log buffer in the SGA.
MAX_DUMP_FILE_SIZE	Maximum size of the trace files, specified as number of operating system blocks.
PROCESSES	Maximum number of operating system processes that can connect simultaneously to this instance.
SQL_TRACE	Enables or disables the SQL trace facility for every user session.
TIMED_STATISTICS	Enables or disables timing in trace files and in monitor screens.

Stages in Startup and Shutdown



Starting Up in Stages

When starting the database, you choose the state in which it starts.

The following scenarios describe different stages of starting up an instance.

Starting the Instance Usually you would start an instance without mounting a database only during database creation or the re-creation of control files.

Starting an instance includes the following tasks:

- Reading the parameter file `initSID.ora`
- Allocating the SGA
- Starting the background processes
- Opening the ALERT file and the trace files

The database must be named with the `DB_NAME` parameter either in the `initSID.ora` file or in the `STARTUP` command.

Starting Up in Stages (continued)

Mounting the Database To perform specific maintenance operations, you start an instance and mount a database but do not open the database.

For example, the database must be mounted but not open during the following tasks:

- Renaming data files
- Enabling and disabling redo log archiving options
- Performing full database recovery

Mounting a database includes the following tasks:

- Associating a database with a previously started instance
- Locating and opening the control files specified in the parameter file
- Reading the control files to obtain the names and status of the data files and redo log files (However, no checks are performed to verify the existence of the data files and online redo log files at this time.)

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

Opening the database includes the following tasks:

- Opening the online data files
- Opening the online redo log files

If any of the data files or online redo log files are not present when you attempt to open the database, the Oracle server returns an error.

During this final stage, the Oracle server verifies that all the data files and online redo log files can be opened and checks the consistency of the database. If necessary, the System Monitor background process (SMON) initiates instance recovery.

Instance Recovery

An instance failure occurs when the instance cannot continue to work.

For example, if there is an operating system crash, the SMON background process automatically performs instance recovery when the database is reopened. That is, the online redo log file is used to recover the committed data in the database buffer cache that was lost because of the instance failure.

Instance recovery consists of the following steps:

- 1 Rolling forward to recover data that has not been recorded in the data files but that has been recorded in the online redo log

Instance Recovery (continued)

- 2** Opening the database instead of waiting for all transactions to be rolled back before making the database available (Any data that is not locked by unrecovered transactions is immediately available.)
- 3** Rolling back uncommitted transactions by SMON and by the individual server processes as they access locked data

Shutting Down in Stages

There are three steps to shutting down an instance and the database to which it is connected.

Closing the Database The first step in shutting down a database is closing the database. When the database is closing, the Oracle server writes the buffer cache changes and redo log buffer cache entries to the data files and online redo log files. After this operation, the Oracle server closes all online data files and online redo log files. The control files remain open while a database is closed but still mounted.

Dismounting the Database The second step is dismounting the database from an instance. After you dismount a database, only an instance remains.

When a database is dismounted, the Oracle server closes its control files.

Shutting Down the Instance The final step in database shutdown is shutting down the instance. When you shut down an instance, the ALERT file and the trace files are closed, the SGA is deallocated, and the background processes are terminated.

Starting Up the Instance

STARTUP Command

Start up the instance and open the database:

```
STARTUP PFILE=/DISK1/initU15.ora
```

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE™

Starting Up

To start up an instance, use the following command:

```
STARTUP [FORCE] [RESTRICT] [PFILE=filename]  
        [OPEN] [RECOVER] [database]  
        | MOUNT  
        | NOMOUNT]
```

Note: This is not the complete syntax.

where:

OPEN	enables users to access the database
MOUNT	mounts the database for certain DBA activities but does not allow user access to the database
NOMOUNT	creates the SGA and starts up the background processes but does not allow access to the database
PFILE= <i>parfile</i>	allows a nondefault parameter file to be used to configure the instance

FORCE	aborts the running instance before performing a normal startup
RESTRICT	enables only users with RESTRICTED SESSION privilege to access the database
RECOVER	begins media recovery when the database starts

Technical Note

With Oracle8i, all Server Manager commands, such as STARTUP and SHUTDOWN, have been merged into SQL*Plus. Throughout this course, all the syntax examples will use SQL*Plus, not Server Manager.

Server Manager is still being shipped with Oracle8i for backward compatibility. Server Manager line mode can be started using the SVRMGRL executable.

Automating Database Startup

On Windows NT The Oracle database on Windows NT runs as a service, which avoids process termination when a user logs out. A service must be registered by the service subsystem of Windows NT.

To start the Oracle services at the startup time, use the control panel in the Service dialog box and select Automatic as the startup type.

Before Oracle8i The database can be opened by starting the following two services:

- OracleServiceSID: Is created for the database instance SID
- OracleStartSID: Starts the database automatically by running the `strtSID.cmd` script

Automating Database Startup (continued)

With Oracle8i The database can be opened by starting the following service:

`OracleServiceSID`: Is created for the database instance `SID`

To start the database automatically, you will have to make sure that the parameter `ORA_SID_AUTOSTART` is set to `True` in the registry. (For more information, refer to the installation guide for your operating system.)

On UNIX On UNIX, automating database startup and shutdown can be controlled by the entries in a special operating system file; for example, `oratab` in the `/var/opt/oracle` directory. (For more information, refer to the installation guide for your operating system.)

Troubleshooting

Attempting to start the Oracle Utilities without starting these services results in one of the following error messages:

```
ORA-12547: TNS: lost contact
```

or

```
ORA-09352: Windows 32-bit two-task driver unable to spawn new  
Oracle task
```

Changing Database Availability

ALTER DATABASE Command

- Change the state of the database from NOMOUNT to MOUNT:

```
ALTER DATABASE database MOUNT;
```

- Open the database as a read-only database:

```
ALTER DATABASE database OPEN READ ONLY;
```

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE

Changing the Status of the Database

To open the database from STARTUP NOMOUNT to a MOUNT stage or from MOUNT to an OPEN stage, use the ALTER DATABASE command:

```
ALTER DATABASE { MOUNT | OPEN }
```

To prevent data from being modified by user transactions, the database can be opened in read-only mode.

To start up an instance, use the following command:

```
ALTER DATABASE OPEN  
[READ WRITE | READ ONLY]
```

where:

READ WRITE	opens the database in read-write mode, allowing users to generate redo logs
READ ONLY	restricts users to read-only transactions, preventing them from generating redo log information

Opening a Database in Read-Only Mode

Opening a Database in Read-Only Mode

- Any database can be opened as a read-only database
- A read-only database can be used to:
 - Execute queries
 - Execute disk sorts using locally managed tablespaces
 - Take data files offline and online, not tablespaces
 - Perform recovery of offline data files and tablespaces

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE[®]

Read-Only Database Features

Any database can be opened as read-only, as long as it is not already open in READ WRITE mode. The feature is especially useful for a standby database to offload query processing from the production database.

If a query needs to use temporary tablespace—for example, to do disk sorts—the current user must have a locally managed tablespace assigned as the default temporary tablespace; otherwise, the query will fail. For user SYS, a locally managed tablespace is required.

Note: Locally managed tablespaces will be discussed in a later lesson.

Read-only mode does not restrict database recovery or operations that change the database state without generating redo data. For example, in read-only mode:

- Data files can be taken offline and online
- Recovery of offline data files and tablespaces can be performed

Disk writes to other files, such as control files, operating system audit trails, trace files, and ALERT files, can continue in read-only mode.

Shutting Down

Shutdown Options				
Shutdown Mode	A	I	T	N
Allow new connections	x	x	x	x
Wait until current sessions end	x	x	x	o
Wait until current transactions end	x	x	o	o
Force a checkpoint and close files	x	o	o	o

Shutdown mode:

A Abort I Immediate

x
o

 NO

T Transactional N Normal YES

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE

Shutting Down the Database

Shut down the database to make operating system offline backups of all physical structures and to modify initialization parameters.

To shut down an instance, use the following command:

```
SHUTDOWN [NORMAL | TRANSACTIONAL | IMMEDIATE | ABORT ]
```

Shutdown Normal

Normal is the default shutdown mode. Normal database shutdown proceeds with the following conditions:

- No new connections are allowed.
- The Oracle server waits for all users to disconnect before completing the shutdown.
- Oracle closes and dismounts the database before shutting down the instance.
- The next startup will not require an instance recovery.

Shutdown Transactional

A transactional shutdown prevents clients from losing work. A transactional database shutdown proceeds with the following conditions:

- No client can start a new transaction on this particular instance.
- A client is disconnected when the client ends the transaction that is in progress.
- When all transactions have finished, a shutdown immediate occurs.
- The next startup will not require an instance recovery.

Shutdown Immediate

Immediate database shutdown proceeds with the following conditions:

- Current SQL statements being processed by Oracle are not completed.
- The Oracle server does not wait for users currently connected to the database to disconnect.
- Oracle rolls back active transactions and disconnects all connected users.
- Oracle closes and dismounts the database before shutting down the instance.
- The next startup will not require an instance recovery.

Shutdown Abort

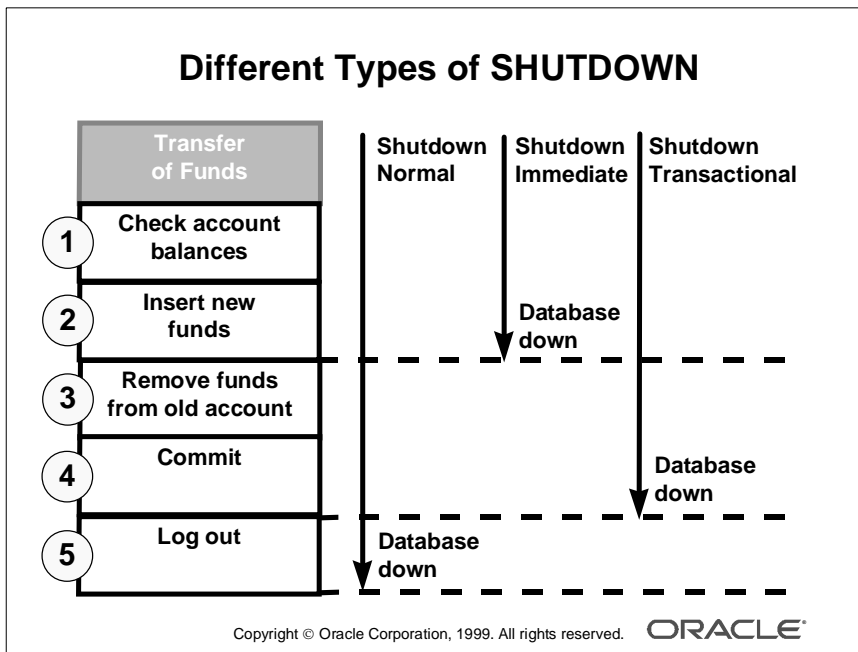
If the normal and immediate shutdown options do not work, you can abort the current database instance. Aborting an instance proceeds with the following conditions:

- Current SQL statements being processed by the Oracle server are immediately terminated.
- Oracle does not wait for users currently connected to the database to disconnect.
- Uncommitted transactions are not rolled back.
- The instance is terminated without closing the files.
- The next startup will require instance recovery.

Note: On Windows NT before Oracle8i, you can close the database by stopping the following two services: `OracleServiceSID` and `OracleStartSID`.

By stopping the `OracleServiceSID` service, the `OracleStartSID` is also terminated and the `orashut.bat` script is executed.

On Windows NT with Oracle8i, you can close the database by stopping the `OracleServiceSID` service and setting the `ORA_SID_SHUTDOWN` parameter to `True` in the registry. (For more information, refer to the installation guide for your operating system.)



Sequence of Events During Different Types of SHUTDOWN

The slide shows the sequence of events when the different SHUTDOWN commands are entered after step 1 is executed.

Steps 1 through 5 describe a transfer of funds from one bank account to another.

- 1 Query the accounts to check the account balances.
- 2 Execute an `INSERT` command to transfer the funds to the new bank account.
- 3 Execute a `DELETE` command to remove the funds from the old bank account.
- 4 Execute a `COMMIT` to finish the transaction successfully.
- 5 Disconnect from the Oracle server.

With a normal shutdown, the Oracle server waits for all users to disconnect before completing the shutdown.

When an immediate shutdown occurs, the Oracle server terminates the current SQL command in step 2 and rolls back the active transaction.

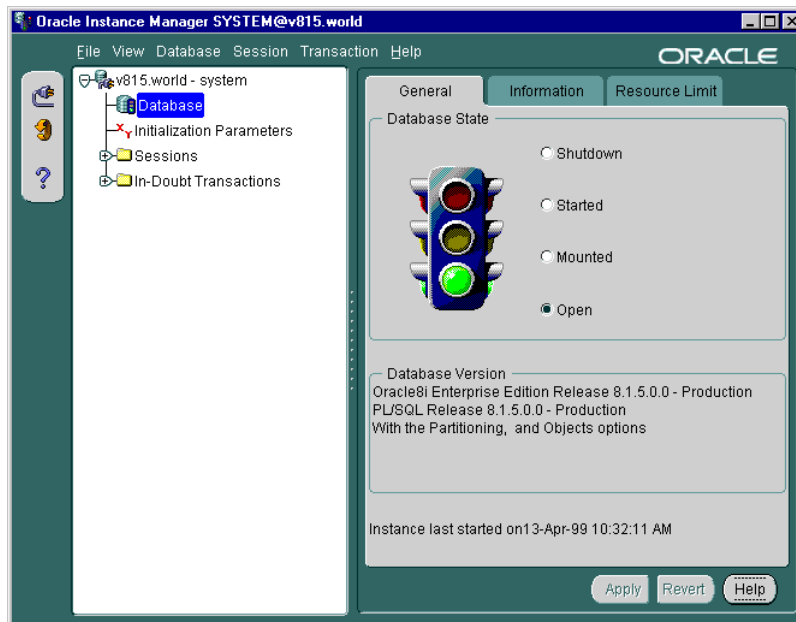
With a transactional shutdown, the Oracle server waits until step 4 is processed—that is, until the transaction is completed—then an immediate shutdown occurs.

When an abort shutdown occurs, the Oracle server terminates the current SQL command, but the active transaction is not rolled back.

How to Use Instance Manager to Start Up or Shut Down

Launch Instance Manager to start up or shut down a database.

- 1 Launch Instance Manager and connect directly to the database:
Start—>Programs—>Oracle - *EMV2 Home*—>DBA Management Pack
—>Instance Manager
- 2 Enter the login information, and click OK.



- 3 Choose the state of your database by selecting a database state option button, and click Apply.
- 4 Depending on the selection, you will have to choose the stage before clicking OK to execute the operation.

Getting and Setting Parameter Values

Dynamic Performance Views

- Are maintained by the Oracle server and continuously updated
- Contain data about disk and memory structures
- Contain data that is useful for performance tuning
- Have public synonyms with the prefix V\$

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE

Dynamic Performance Views

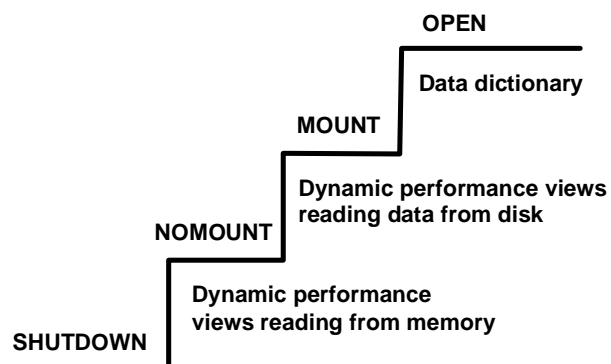
These views are called *dynamic performance views* because they are continuously updated while a database is open and in use. Their contents relate primarily to performance. They provide data about internal disk structures and memory structures and are accessible to the database administrator.

Dynamic performance views are identified by the prefix V_\$, but Oracle provides public synonyms with the prefix V\$.

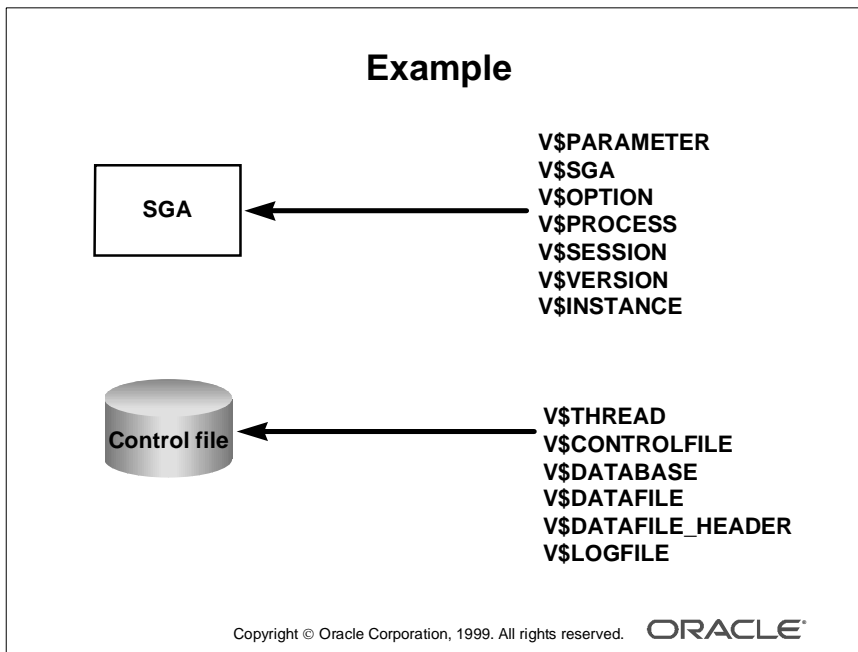
Once the instance is started in the NOMOUNT stage, V\$ views that can be read from memory are accessible. Views that read data from the control file require that the database be mounted.

The V\$FIXED_TABLE view displays all dynamic performance views.

Accessing Dynamic Performance Views



Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**



Example

Dynamic Performance View (accessible in the NOMOUNT stage)	Description
V\$PARAMETER	Contains information about the initialization parameters
V\$SGA	Contains summary information on the SGA
V\$OPTION	Lists options that are installed with the Oracle server
V\$PROCESS	Contains information about the currently active processes
V\$SESSION	Lists current session information
V\$VERSION	Lists the version number and the components
V\$INSTANCE	Displays the state of the current instance

Example (continued)

Dynamic Performance View (accessible in the MOUNT stage)	Description
V\$THREAD	Contains thread information; for example, about the redo log groups
V\$CONTROLFILE	Lists the names of the control files (Even though available, this view returns no rows in NOMOUNT state.)
V\$DATABASE	Contains database information
V\$DATAFILE	Contains data file information from the control file
V\$DATAFILE_HEADER	Displays data file header information from the control file
V\$LOGFILE	Contains information about the online redo log files

Displaying Current Parameter Values

- Use the command:

```
SHOW PARAMETER control
```

- Query the V\$PARAMETER dynamic performance view:

```
SELECT name FROM v$parameter  
WHERE name LIKE '%control%';
```

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Displaying Current Parameter Values

To determine the parameter settings for a database that has been started, use the SHOW PARAMETER command.

This command displays all parameters in an alphabetical order with their current values.

Enter the following text string to display all parameters having CONTROL in their name:

```
SQL> SHOW PARAMETER control
```

NAME	TYPE	VALUE
-----	-----	-----
control_file_record_keep_time	integer	7
control_files	string	/DISK1/control01.con

You can also use the V\$PARAMETER dynamic performance view to determine the current settings of any parameter.

Dynamic Initialization Parameters

- Some initialization parameters can be modified while an instance is running.

```
ALTER SESSION SET SQL_TRACE=true;
```

```
ALTER SYSTEM SET TIMED_STATISTICS=true;
```

```
ALTER SYSTEM SET SORT_AREA_SIZE=131072 DEFERRED;
```

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Change Initialization Parameters Dynamically

Some initialization parameters are dynamic—that is, they can be modified using the ALTER SESSION, ALTER SYSTEM, or ALTER SYSTEM DEFERRED command while an instance is running.

```
ALTER SESSION SET parameter_name = value
```

```
ALTER SYSTEM SET parameter_name = value [DEFERRED]
```

The ALTER SESSION command modifies the value of the parameter only for the session that executes the command.

The ALTER SYSTEM command globally changes the value of the parameter. The new value is effective until shutdown or until it is changed again.

The ALTER SYSTEM DEFERRED command modifies the value of the parameter for future sessions that connect to the database.

Query the V\$PARAMETER or V\$SYSTEM_PARAMETER view to list information about the modified parameter.

```
SQL> SELECT isses_modifiable,issys_modifiable,
2      ismodified, name
3      FROM v$system_parameter
4      WHERE ismodified != 'FALSE';
```

```
ISSES ISSYS_MOD ISMODIFI NAME
```

```
-----
TRUE  IMMEDIATE MODIFIED timed_statistics
```

```
1 row selected.
```

Change Initialization Parameters Dynamically (continued)

The columns display the following information:

- **ISSES_MODIFIABLE**: Indicates whether the parameter can be modified by **ALTER SESSION**
 - **ISSYS_MODIFIABLE**: Indicates whether the parameter can be modified by **ALTER SYSTEM**
 - **ISMODIFIED**: Indicates **ALTER SESSION** is modified with the value **MODIFIED** and **ALTER SYSTEM** is modified with the **SYS_MODIFIED** value
- V\$PARAMETER** shows the current session values and **V\$SYSTEM_PARAMETER** shows the current system values independent of the session. For example, if the **ALTER SYSTEM DEFERRED** command is executed, the **ISMODIFIED** column in the **V\$SYSTEM_PARAMETER** dynamic performance view contains the value **MODIFIED**, but the column in the **V\$PARAMETER** dynamic performance view displays the value **False** in the same session.

Note: The **ALTER SYSTEM** or **ALTER SYSTEM DEFERRED** command that modifies a parameter is recorded in the trace file called the **ALERT** file.

Managing Sessions

Enable and Disable Restricted Session

- Use the **STARTUP** command to restrict access to a database:

```
STARTUP RESTRICT
```

- Use the **ALTER SYSTEM** command to place an instance in restricted mode:

```
ALTER SYSTEM ENABLE RESTRICTED SESSION;
```

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE™

Restricted Session

A restricted session is useful, for example, when you perform structure maintenance or a database export and import. The database can be started in restricted mode so that it is available only to users with the **RESTRICTED SESSION** privilege.

The database can also be put in restricted mode by using the **ALTER SYSTEM SQL** command:

```
ALTER SYSTEM  
{ENABLE|DISABLE}RESTRICTED SESSION
```

where:

ENABLE RESTRICTED SESSION	enables future logins only for users who have the RESTRICTED SESSION privilege
DISABLE RESTRICTED SESSION	disables RESTRICTED SESSION so that users who do not have the privilege can log on

Restricted Session (continued)

Note: The ALTER SYSTEM command does not disconnect current sessions but allows future connections only to users with the RESTRICTED SESSION privilege.

The V\$INSTANCE dynamic performance view contains information about the restricted mode.

```
SQL> SELECT logins FROM v$instance;
LOGINS
-----
RESTRICTED
1 row selected.
```

Terminating Sessions

- Identify which session to terminate with the V\$SESSION dynamic performance view:

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

- Execute the ALTER SYSTEM command:

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

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

How to Terminate Sessions

After placing an instance in restricted mode, you may want to kill all current user sessions before performing administrative tasks.

```
ALTER SYSTEM KILL SESSION 'integer1,integer2'
```

where: KILL SESSION Identifies the session with both of the following values from the V\$SESSION view:
integer1: value of the SID column
integer2: value of the SERIAL# column

Note: The session ID and serial number are used to uniquely identify a session. This guarantees that the ALTER SYSTEM command is applied to the correct session even if the user logs off and a new session uses the same session ID.

Effects of Terminating a Session

The ALTER SYSTEM KILL SESSION command causes the background process PMON to perform the following steps upon execution:

- Roll back the user's current transaction
- Release all currently held table or row locks
- Free all resources currently reserved by the user

Effects of Terminating a Session (continued)

You query the V\$SESSION view to identify the session ID and serial number of user sessions.

Terminating an Active Session If a user session is making an SQL call to the Oracle server—that is, the session is ACTIVE—when it is terminated, the transaction is rolled back and the user immediately receives the following message:

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

If the user session is performing some activity that must be completed and cannot be interrupted, the Oracle server waits for this activity to finish.

Terminating an Inactive Session If the session is inactive when it is terminated, the ORA-00028 message is not returned immediately, but the STATUS column in the V\$SESSION view is marked killed.

When the user attempts to use the terminated session again, the ORA-00028 message is returned and the row for the terminated session is removed from V\$SESSION.

Note: When a session is terminated, the Oracle server does not kill the operating system processes.

However, the following command, normally used in a parallel server environment, disconnects a session when its current transaction is finished and terminates the server process:

```
ALTER SYSTEM DISCONNECT SESSION 'integer1, integer2'  
POST_TRANSACTION
```

The ALERT File and the Trace Files

- **Trace files can be written by server and background processes.**
- **The Oracle server dumps information about errors in trace files.**
- **The ALERT file consists of a chronological log of messages and errors.**
- **Server process tracing can be enabled or disabled by:**
 - **An ALTER SESSION command**
 - **The parameter SQL_TRACE**

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

The ALERT File and the Trace Files

If an error occurs while your Oracle instance is running, the messages are written to the ALERT file. During startup of the database, if the ALERT file does not exist, Oracle creates one.

The ALERT file of a database is a chronological log of messages and errors. Oracle uses the ALERT file as an alternative to displaying such information.

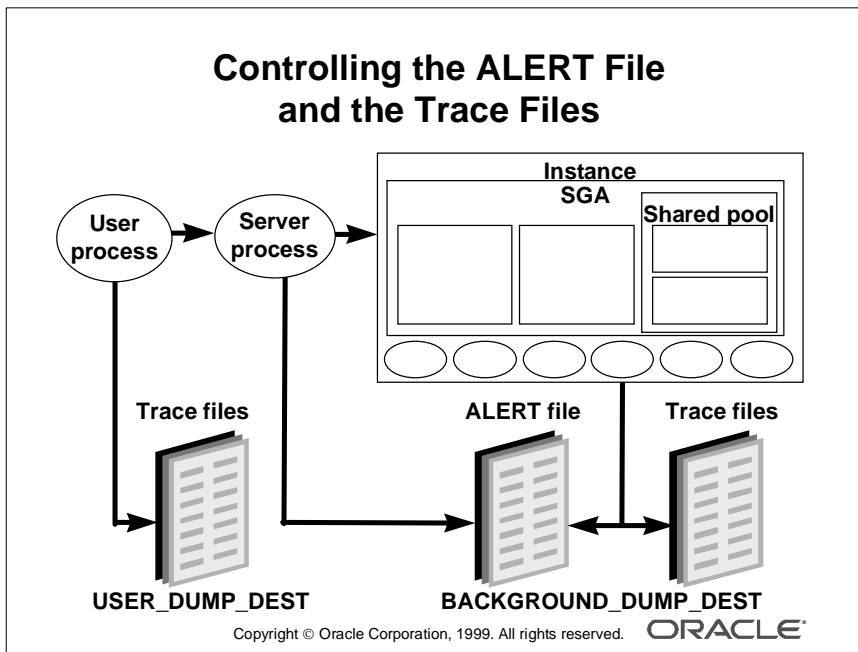
If an error is detected by a background process, the information is dumped into a trace file.

Trace files can also be generated by server processes at user request.

Tracing can be enabled or disabled by the initialization parameter SQL_TRACE; the value is True or False.

The following statement enables writing to a trace file for a particular session:

```
SQL>ALTER SESSION SET sql_trace=TRUE;
```



Location of the ALERT File and the Trace Files

The following parameters control the location and size of the ALERT file and the trace files:

Initialization Parameter	Description
BACKGROUND_DUMP_DEST	Defines the location of the background trace file and ALERT file
USER_DUMP_DEST	Defines where trace files will be created at the request of the users
MAX_DUMP_FILE_SIZE	Specified in O/S blocks; limits the size of user trace files, not the ALERT file or background trace files

Note

- The MAX_DUMP_FILE_SIZE and USER_DUMP_DEST parameters are dynamic initialization parameters.
- On UNIX, the ALERT file is named `alert_SID.log` and is located in the `$ORACLE_HOME/rdbms/log` directory by default.
- On Windows NT, the ALERT file is named `SIDalrt.log` and is by default located in the `%ORACLE_HOME%\RDBMS80\TRACE` directory in Oracle8. If Oracle8i, it is located in the `%ORACLE_HOME%\ADMIN\SID\BDUMP` directory.

Guidelines

Check the ALERT file periodically to:

- **Detect internal errors (ORA-600) and block corruption errors**
- **Monitor database operations**
- **View the nondefault initialization parameter**

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Guidelines for the ALERT File

It is important to check the ALERT file regularly to detect problems before they become serious.

The following information is logged in the ALERT file:

- All internal errors (ORA-00600) and block corruption errors (ORA-01578)
- Operations that affect database structures and parameters, as well as commands, such as STARTUP, SHUTDOWN, ARCHIVE LOG, and RECOVER
- The values of all nondefault initialization parameters at the time the instance starts

Summary

Summary

In this lesson, you should have learned how to:

- Create the parameter file
- Start up and shut down an instance
- Access dynamic performance views
- Manage sessions
- Describe the use of the ALERT file and the trace files

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE®

Quick Reference

Context	Reference
Initialization parameters	DB_NAME CONTROL_FILES SHARED_POOL_SIZE BACKGROUND_DUMP_DEST DB_BLOCK_BUFFERS COMPATIBLE IFILE LOG_BUFFER PROCESSES SQL_TRACE
Dynamic initialization parameters	USER_DUMP_DEST MAX_DUMP_FILE_SIZE TIMED_STATISTICS
Dynamic initialization parameters (deferred)	SORT_AREA_SIZE
Dynamic performance views	V\$FIXED_TABLE V\$PARAMETER V\$CONTROLFILE V\$DATABASE V\$DATAFILE V\$DATAFILE_HEADER V\$INSTANCE V\$LOGFILE V\$OPTION V\$PROCESS V\$PWFILERS V\$SESSION V\$SYSTEM_PARAMETER V\$SGA V\$VERSION

Context	Reference
Data dictionary views	None
Commands	CONNECT / AS SYSDBA CONNECT / AS SYSOPER STARTUP SHUTDOWN SHOW PARAMETER ALTER SYSTEM KILL SESSION ALTER SYSTEM DISCONNECT SESSION ... POST_TRANSACTION ALTER SYSTEM ENABLE RESTRICTED SESSION ALTER SYSTEM DISABLE RESTRICTED SESSION ALTER SESSION SET ALTER SYSTEM SET ALTER SYSTEM SET... DEFERRED ALTER DATABASE MOUNT ALTER DATABASE OPEN ALTER DATABASE OPEN READ ONLY ALTER DATABASE OPEN READ WRITE
Packaged procedure and functions	None

4

Creating a Database

Objectives

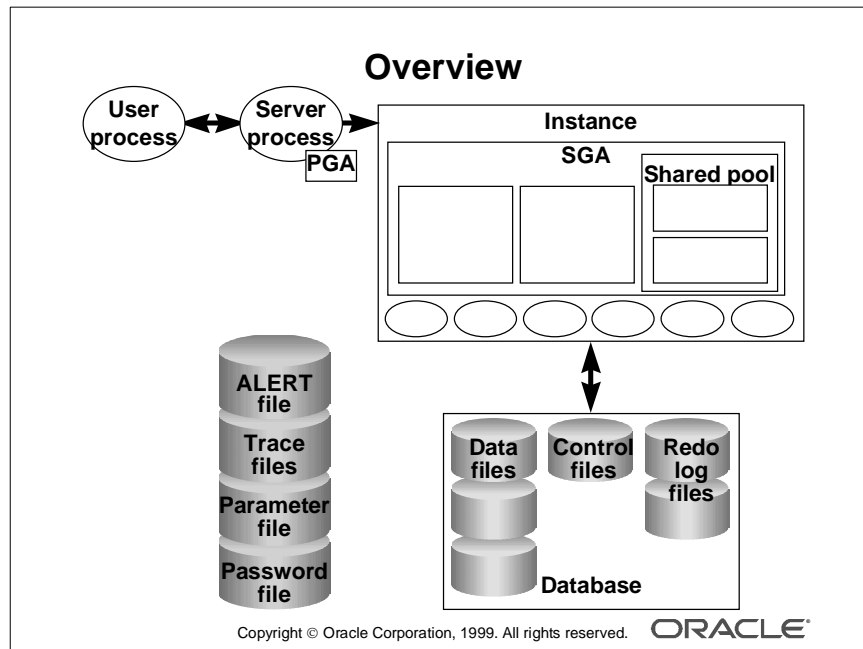
Objectives

After completing this lesson, you should be able to do the following:

- **Prepare the operating system**
- **Prepare the parameter file**
- **Create the database**

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

Overview



Overview of Managing and Organizing a Database

Creating the database is the first step in managing and organizing a database system.

Database creation is a task that prepares several operating system files and is needed only once no matter how many data files the database has. This is a very important task because you must decide on database settings, such as the size of the database block and the database character set, which cannot be changed after the creation.

Depending on the operating system, a database may have been created automatically as part of the installation.

You can use this initial database, or you can erase it and create a new one manually.

During migration from an older version of Oracle, database creation is necessary only if an entirely new database is needed. Otherwise you can use a migration utility—for example, Oracle Data Migration Assistant—to migrate from an earlier version of the database.

You can create a database with new data files, or you can erase information in an existing database that has the same physical structure.

The CREATE DATABASE command initiates the creation of the control files, redo log files, and the data dictionary structure that the Oracle server requires to access the database.

Preparing the Operating System

Creation Prerequisites

- A privileged account authenticated in one of the following ways:
 - By the operating system
 - Using a password file
- Memory to start the instance
- Sufficient disk space for the planned database

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE

Considerations Before Creating a Database

You should be fully privileged on the operating system or should use the password file authentication (see the lesson “Managing an Oracle Instance”).

Before you create the database, make sure that the memory for the SGA, the Oracle executable, and the processes is sufficient. Refer to your operating system installation and administration guides.

Calculate the necessary disk space for the database, including online redo log files, control files, and data files.

Planning Database File Locations

- **Keep at least two active copies of a database control file on at least two different devices.**
- **Multiplex the redo log files and put group members on different disks.**
- **Separate data files whose data:**
 - **Will participate in disk resource contention across different physical disk resources**
 - **Have different life spans**
 - **Have different administrative characteristics**

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Planning File Location

Plan how to protect the database, including the online redo log files, control files, data files, and archived redo log files, and provide a backup strategy.

Control Files For the sake of safety, you should create at least two control files on two different disks (see the lesson “Maintaining the Control File”). Because control file copies must always be placed on different disks, they can have identical names, such as `control01.ctl` on UNIX or `control.ora` on Windows NT.

Online Redo Log Files The online redo log files of a database should consist of multiplexed groups of online redo log files. A group of log files consists of identical copies, which should be located on different disks (see the lesson “Maintaining Redo Log Files”).

To distinguish between groups and their members, use a name such as `log0101.rdo` or `log01a.rdo`.

Planning File Location (continued)

Data Files Name data files by relating to the contents as the root of the name—for example, data files such as `system01.dbf`, `temp01.dbf`, and `users01.dbf` on UNIX and `system01.ora` and `temp01.ora` on Windows NT.

Consider the characteristics of the data to be stored before determining the structure appropriate for your database, in order to:

- Minimize fragmentation
- Minimize disk contention
- Separate objects

To minimize fragmentation of the database, you should separate database objects with different life spans, such as application data and temporary data, into different tablespaces.

To ensure well-balanced I/O loads, you should separate objects with competing I/O requirements, such as tables and indexes, into different tablespaces.

Note: These subjects are covered in detail in the lessons “Maintaining Tablespaces and Data Files” and “Storage Structure and Relationships.”

Oracle Software Locations

On UNIX

```
/oracle_base
  /product
    /release_number
      /bin
      /dbs
      /orainst
      /sqlplus
      ...
    /8.1.5
  /admin
/local
```

On Windows NT

```
\oracle_base
  \oracle_home
    \admin\db_name
    \admin\v815
    \oradata\db_name
    ...
  \bin
  \database
```

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

Oracle Database Files

```
/u02/  
  oradata/  
    db01/  
      system01.dbf  
      control01.ctl  
      redo0101.rdo  
      ...  
    db02/  
      system01.dbf  
      control01.ctl  
      redo0101.rdo  
      ...
```

```
/u03/  
  oradata/  
    db01/  
      tools01.dbf  
      control02.ctl  
      redo0102.rdo  
      ...  
    db02/  
      users01.dbf  
      control02.ctl  
      redo0102.rdo  
      ...
```

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Optimal Flexible Architecture

Another important issue during installation and creation of a database is organizing the file system so that it is easy to administer growth by adding data into an existing database, adding users, creating new databases, and adding hardware and distributing I/O load across sufficiently many drives.

The Optimal Flexible Architecture (OFA) standard, which provides one solution to these issues, was written by an Oracle team responsible for installing, tuning, and upgrading UNIX systems. OFA facilitates configuration of complex Oracle systems with low maintenance. During an Oracle installation, the OFA standard configuration is used automatically.

The OFA structure:

- `ORACLE_BASE` is the root of the directory tree.
- `ORACLE_HOME` is the subdirectory that contains the Oracle software and data. Each version has a separate home.
- Distinguish between product files, which consist of Oracle server software and tools; administrative files such as database creation scripts; initialization scripts; and local software that is used with the Oracle server. In the UNIX example, the directories `product`, `admin`, and `local` satisfy this requirement.
- Make a directory explicitly for storing Oracle server data at the same level of each of the devices, such as `ORACLE_HOME/oradata`.
- Make a directory beneath the Oracle directory for each of the databases on the system. In the example, the databases are named `dba01` and `db01`.

Note: The OFA directory structure for NT is not identical to that for UNIX because of the naming conventions and the lack of symbolic links. To circumvent this problem, the creation of hard directories instead of symbolic links such as `u01` and `u02` is necessary.

For example, instead of creating `/u01/oradata/db01/`, where `u01` represents a mount point on UNIX, you would create a directory `DISK_3:\ORADATA\DB01\` on Windows NT.

(For more details, see your installation guide for your platform.)

Creating a Database

Creating a Database

- Created using the Database Configuration Assistant
- Created manually using the **CREATE DATABASE** command

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE[®]

Methods for Creating a Database

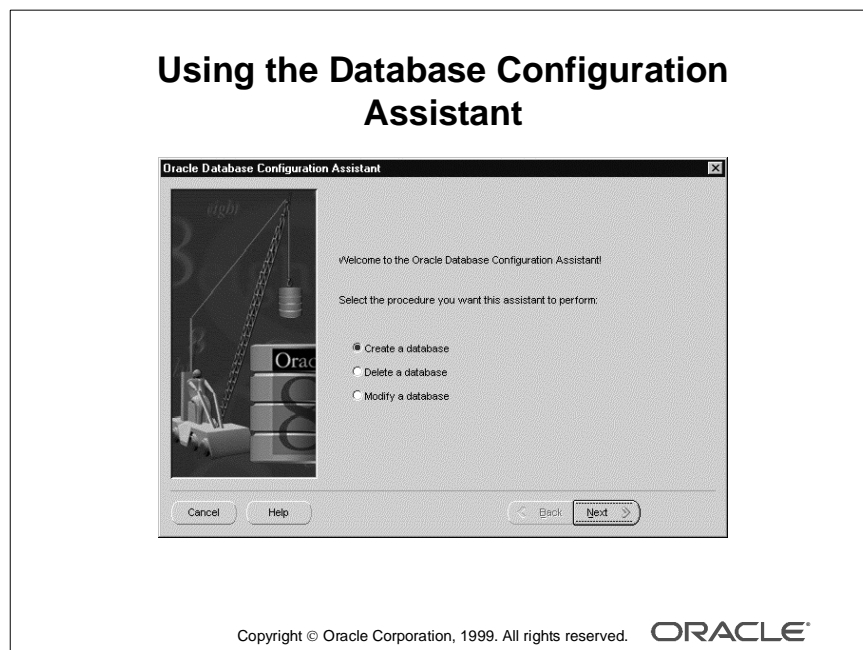
Creating a database can be done either by using the Database Configuration Assistant or by creating a SQL script using the **CREATE DATABASE** command. The Database Configuration assistant is Java-based and can be launched from any platform with a Java engine.

During the installation you are prompted to create a database with the Database Configuration Assistant. This utility can also be used after installation to create or to delete a database.

Technical Note

Before release 8.1, the Database Configuration Assistant was available only on Windows NT. To create a database on UNIX, the installer would prompt you to enter the number and the location of the mount points, the character set, the national character set, passwords for user **SYS** and **SYSTEM**, and the UNIX group password to enable operating system authentications.

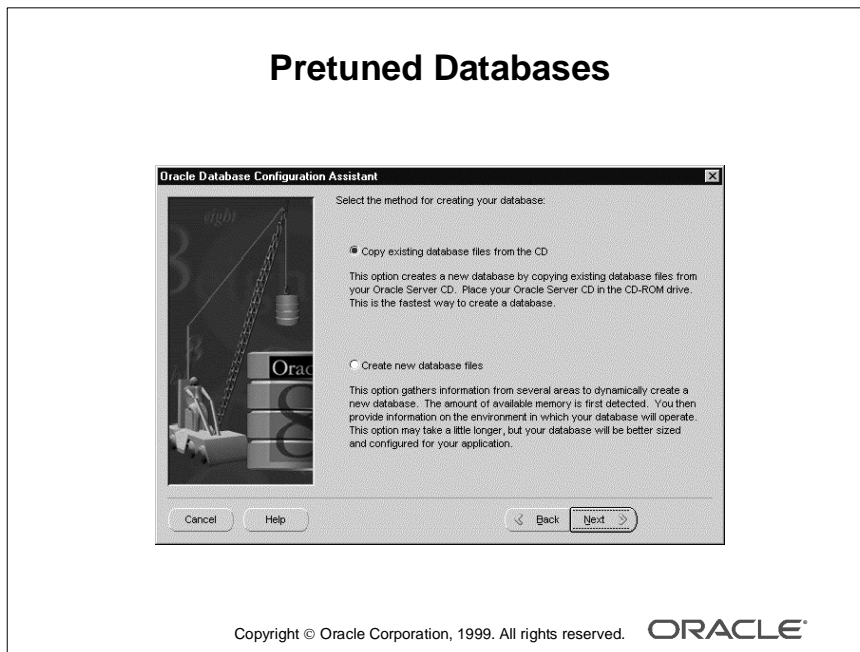
Using the Database Configuration Assistant



How to Start the Database Configuration Assistant

To start the Database Configuration Assistant after the installation, follow these steps:

- 1 Select Start—>Programs—>Oracle -*ORACLE_HOME*—>Database Administration—>Database Configuration Assistant.
This utility is the simplest method to create an Oracle database.
- 2 Go to the last Database Configuration Assistant screen and complete the creation. The Database Configuration Assistant starts the related services, edits the `init.ora` files, and creates a database, configures options, or deletes the database and services.

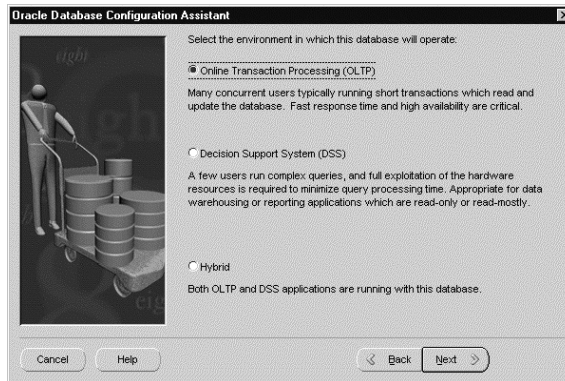


Choosing a Pretuned Database or Customizing a Database

The Database Configuration Assistant offers different options:

- *Typical* consists of two suboptions:
 - Copy existing database files from the CD: Automatically installs a standard database with default initialization parameter settings
 - Create new database files: Asks you several database environment questions before dynamically creating the database
- *Custom* enables you to customize the creation of your database. This option is only for database administrators experienced with advanced database creation procedures, such as customizing:
 - Data, control, and redo settings
 - Tablespace sizes
 - Extent sizes
 - Database memory parameters
 - Archiving formats and destinations
 - Trace file destinations
 - Character set values

Predefined Sample Schemas



Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE®

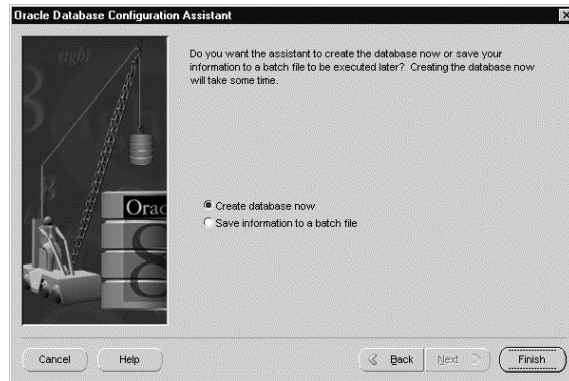
Predefined Samples Schemas

You can use either the Typical option or the Custom option to specify the type of environment in which to operate your database:

Environment	Description
Online Transaction Processing (OLTP)	Databases in OLTP environments must process thousands or even millions of transactions from many concurrent users each day. These transactions consist of reading, writing, and deleting data in the database. Users must have quick access to most current data. Therefore, database performance is defined in terms of throughput and availability of data.
Decision Support System (DSS)	Databases in DSS environments must process a variety of queries (typically read-only), ranging from a simple fetch of a few records to numerous complex queries that sort thousands of records from many different tables. Therefore, database performance is defined in terms of response time.
Hybrid	Hybrid databases support both OLTP and DSS environments.

Sample OLTP and DSS database schemas are available for the Oracle8i Enterprise Edition.

Finishing Create Database



Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE®

Finishing the Database Configuration Assistant

You can use the Database Configuration Assistant to either create the database now or save the information into a batch file and execute it later.

Creating a Database Manually

Creating a Database Manually

1. Decide on a unique instance and database name and database character set.
2. Set the operating system variables.
3. Prepare the parameter file.
4. Create a password file (recommended).
5. Start the instance.
6. Create the database.
7. Run scripts to generate the data dictionary and accomplish postcreation steps.

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE®

Steps for Creating a Database Manually

This lesson covers the first six steps in detail.

The last step, generating the data dictionary, is discussed in the lesson “Data Dictionary Views and Standard Packages.”

Technical Note

See your operating system–specific Oracle documentation for information about creating databases on your platform.

Operating System Environment

On UNIX, set the following environment variables:

- ORACLE_HOME
- ORACLE_SID
- ORACLE_BASE
- ORA_NLS33
- PATH

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Setting the Operating System Environment

Decide on a unique name for the instance and set the following environment variables:

Variable	Description
ORACLE_HOME	Specifies the directory where the Oracle software will be installed (Example: /u01/app/oracle/product/8.1.5)
ORACLE_SID	Specifies the instance name and must be unique for Oracle instances running on the same machine.
ORACLE_BASE	Not required, but recommended as part of an OFA-compliant installation (Example: /u01/app/oracle)
ORA_NLS33	Required when creating a database with a character set other than US7ASCII (Example: \$ORACLE_HOME/ocommon/nls/admin/data)
PATH	Search path, which must include \$ORACLE_HOME/bin

For example, set the environment in a Korn shell with the following command:

```
$ORACLE_SID=U16; export ORACLE_SID
```

In a C shell, execute the following command:

```
$setenv ORACLE_SID U16
```

Technical Note

- If ORA_NLS is not set and the database is started with other languages and character sets than the database default, they will not be recognized.
- The value of the SID can be up to eight characters. Before 8.1 it could only be up to four characters on Windows NT.

Operating System Environment

On Windows NT:

- Set the variable `ORACLE_SID` to use SQL*Plus.
- Create the service and the password file with `ORADIM`.

```
C:\> ORADIM -NEW -SID u16 -INTPWD password  
-STARTMODE auto  
-PFILE ORACLE_HOME\DATABASE\initU16.ora
```

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE®

Setting the Operating System Environment on Windows NT

On Windows NT, Oracle uses variables in the registry similar to the way Oracle on UNIX uses shell environment variables.

The Oracle Installer, the `ORADIM`, and the Database Configuration Assistant utilities define variables in the registry as well as registering the Oracle instance as a service. You can edit the registry manually through the `regedit.exe` utility.

For example, parameters such as `ORACLE_HOME`, `ORA_NLS33`, and `ORACLE_SID` (the default is `ORCL` for the starter database) are stored in the `HKEY_LOCAL_MACHINE\SOFTWARE\ORACLE` folder.

Therefore, the creation of a new database requires `ORACLE_SID` to be set with the following command:

```
C:\> set ORACLE_SID=U16
```

Now you create a new service and the new password file, if required, to run the database with the `ORADIM` utility:

```
C:\>ORADIM -NEW -SID sid [-INTPWD internal_pwd][SRVC svrcname]  
[MAXUSERS number][STARTMODE auto,manual][-PFILE filename]
```

Note: The Oracle server records all operations that are executed with the `ORADIM` utility in the `ORACLE_HOME\DATABASE\ORADIM.LOG` file.

Preparing the Parameter File

- Create the new `initSID.ora`.

```
$cp init.ora $ORACLE_HOME/dbs/initU16.ora
```

- Modify the `initU16.ora` by editing the parameters.

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE®

Editing the Parameter File

```
db_name = U10
instance_name = U10
service_names = U10
db_files = 1024
control_files =
('C:\ORA815\oradata\U10\control01.ctl',
'C:\ORA815\oradata\U10\control02.ctl')
db_file_multiblock_read_count = 8
db_block_buffers = 2048
shared_pool_size = 4194304
log_checkpoint_interval = 10000
log_checkpoint_timeout = 1800
processes = 50
parallel_max_servers = 5
log_buffer = 32768
...
```

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

Preparing the Parameter File

When preparing the new database, copy the default `init.ora` file with the new name `initSID.ora`.

Change the settings for some parameters; others can be left to default.

You should specify at least the following parameters before starting the instance:

Parameter	Description
DB_NAME	Database identifier of eight characters or fewer. This is the only parameter that is required when creating a new database. This parameter does not need to match the ORACLE_SID but must match the name used in the CREATE DATABASE statement.
CONTROL_FILES	Specifies a list of control files. Always specify at least two control filenames, placed on separate disks if possible. The control files do not need to exist at this point. The Oracle server can create new operating system files when creating the database.
DB_BLOCK_SIZE	Determines the database block size. The block size cannot be changed after database creation.

Note: The database name is associated with a database at create database time and is stored in the control files. To change the name of an existing database, use the CREATE CONTROLFILE command to re-create the control file (see the course *Enterprise DBA Part 1B: Backup and Recovery Workshop*).

Starting the Instance

1. Connect as SYSDBA.
2. Start the instance in NOMOUNT stage.

```
SQL> STARTUP NOMOUNT PFILE=initU16.ora  
ORACLE instance started.
```

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Starting the Instance

Connect as SYSDBA using operating system authentication or the password file method and start the instance using the STARTUP command.

The password is the one that was previously used to create the service with the ORADIM (only Windows NT) or ORAPWD utility.

If the parameter file is not in the default location, you may need to specify the PFILE clause in the STARTUP command. For this course, the parameter file is located in the current directory.

Creating the Database

```
SPOOL creU16.log
STARTUP NOMOUNT PFILE=initU16.ora
CREATE DATABASE U16
    MAXLOGFILES 5
    MAXLOGMEMBERS 5
    MAXDATAFILES 100
    MAXLOGHISTORY 100
LOGFILE
    GROUP 1 ('/DISK3/log1a.rdo',/DISK4/log1b.rdo') SIZE 1 M,
    GROUP 2 ('/DISK3/log2a.rdo',/DISK4/log2b.rdo') SIZE 1 M
DATAFILE
    '/DISK1/system01.dbf' size 50M autoextend on
CHARACTER SET WE8ISO8859P1;
```

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

The CREATE DATABASE Command

To create a database, use the following SQL command:

```
CREATE DATABASE [database]
    [CONTROLFILE REUSE]
    [LOGFILE [GROUP integer] filespec
    [, [GROUP integer] filespec]...]
    [MAXLOGFILES integer]
    [MAXLOGMEMBERS integer]
    [MAXLOGHISTORY integer]
    [MAXDATAFILES integer]
    [MAXINSTANCES integer]
    [ARCHIVELOG | NOARCHIVELOG]
    [CHARACTER SET charset]
    [NATIONAL CHARACTER SET charset]
```

The CREATE DATABASE Command (continued)

```
[DATAFILE filespec [autoextend_clause]
[,          filespec [autoextend_clause]...]]

filespec ::= 'filename' [SIZE integer][K|M] [REUSE]

autoextend_clause ::=
[AUTOEXTEND {OFF
              |ON [NEXT integer[K|M]]
              [MAXSIZE {UNLIMITED|integer[K|M]}}]
]
]
```

where:

database	is the name of the database to be created (If the name of the database is omitted, the initialization parameter DB_NAME is used.)
CONTROLFILE REUSE	specifies that an existing control file identified in the parameter file should be reused
LOGFILE GROUP	specifies the names of the log files to be used and the group to which they belong
MAXLOGFILES	specifies the maximum number of redo log file groups that can ever be created for the database
MAXLOGMEMBERS	specifies the maximum number of log file members for a log file group
MAXLOGHISTORY	specifies the maximum number of archived redo logs for automatic media recovery of the Oracle Parallel Server
DATAFILE filespec	specifies the data files to be used
AUTOEXTEND	enables or disables the automatic extension of a data file (see “Maintaining Tablespaces and Data Files”)

The CREATE DATABASE Command (continued)

MAXDATAFILES	specifies the initial sizing of the data file section of the control file at CREATE DATABASE or CREATE CONTROLFILE time. An attempt to add a new file whose number is greater than MAXDATAFILES, but less than or equal to DB_FILES, causes the control file to expand automatically so that the data files section can accommodate more files
MAXINSTANCES	is the maximum number of instances that can simultaneously mount and open the database
ARCHIVELOG	establishes that redo logs must be archived before they can be reused
NOARCHIVELOG	establishes that redo logs can be reused without archiving their contents
CHARACTER SET	is the character set the database uses to store data
NATIONAL CHARACTER SET	specifies the national character set used to store data in columns defined as NCHAR, NCLOB, or NVARCHAR2 If not specified, the national character set is the same as the database character set (see the lesson “Using National Language Support”).

If REUSE is specified in a file specification, then the file must exist; otherwise the SIZE option must be specified and the file must not exist.

Example

Turn on spooling to save messages and run the CREATE statement.

The creation results in a database with the name U16. The database consists of two online redo log file groups, each with two 1 MB members and one 50 MB data file. The database stores data with an 8-bit character set.

The CREATE DATABASE Command (continued)

Note:

- The Oracle server allocates as much space in the control files as the values of MAXLOGMEMBERS, MAXLOGFILES, MAXDATAFILES, MAXLOGHISTORY, and MAXINSTANCES require.
To change the value of these parameters, use the CREATE CONTROLFILE command to re-create the control file. (See the course *Enterprise DBA Part 1B: Backup and Recovery*.)
- There is no DROP DATABASE command. To delete a database, you must delete the data files from the operating system. Use the following query to list the physical data files you will have to remove manually from the operating system:

```
SQL> SELECT name FROM v$datafile
       2  UNION
       3  SELECT name FROM v$controlfile
       4  UNION
       5  SELECT member FROM v$logfile;
NAME
-----
C:\ORA815\ORADATA\V815\CONTROL01.CTL
C:\ORA815\ORADATA\V815\CONTROL02.CTL
C:\ORA815\ORADATA\V815\INDX01.DBF
C:\ORA815\ORADATA\V815\OEMREP01.DBF
C:\ORA815\ORADATA\V815\RBS01.DBF
C:\ORA815\ORADATA\V815\REDO01.LOG
C:\ORA815\ORADATA\V815\REDO01A.LOG
C:\ORA815\ORADATA\V815\REDO01B.LOG
C:\ORA815\ORADATA\V815\REDO02.LOG
C:\ORA815\ORADATA\V815\REDO03.LOG
C:\ORA815\ORADATA\V815\SYSTEM01.DBF
C:\ORA815\ORADATA\V815\TEMP01.DBF
C:\ORA815\ORADATA\V815\USERS01.DBF
13 rows selected.
```

With the Database Configuration Assistant, it is possible to remove the services as well as the data files.

The CREATE DATABASE Command (continued)

- To make the new database the default database on Windows NT, change the ORACLE_SID in the registry.
- It is not possible to change the character set or the national character set after creating the database.
- On Windows NT you can use the `build_db.sql` script located in the `%ORACLE_HOME%\RDBMS\ADMIN` directory to create a database.

Troubleshooting

Creation of the database fails if:

- **There are syntax errors in the SQL script**
- **Files that should be created already exist**
- **Operating system errors such as file or directory permission or insufficient space errors occur**

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

Troubleshooting

If one of the three problems shown on the slide occurs, the CREATE DATABASE statement fails.

In every case, you should shut down the database, delete any files created by the CREATE DATABASE statement, correct the errors, and attempt to create again.

After the Database Is Created

The database contains:

- Data files that make up the **SYSTEM** tablespace
- Control files and redo log files
- The user **SYS** with the password **CHANGE_ON_INSTALL**
- The user **SYSTEM** with the password **MANAGER**
- The rollback segment **SYSTEM**
- Internal tables (but no data dictionary views)

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

After Creating the Database

After the database is created, the database is opened, the SQL script `sql .bsq` is successfully executed, and the database objects named in this slide are created.

You can view the dynamic performance views such as `V$LOGFILE`, `V$CONTROLFILE`, and `V$DATAFILE`, but no data dictionary views are created.

The following lessons explain how to create the data dictionary views to create additional tablespaces, alter and add redo log files, add control files, and so on.

Summary

Summary

In this lesson, you should have learned how to:

- Plan the database structure
- Prepare the operating system environment
- Create the database

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

Quick Reference

Context	Reference
Initialization parameters	DB_NAME CONTROL_FILES DB_BLOCK_SIZE
Dynamic performance views	None
Data dictionary views	None
Commands	CREATE DATABASE
Packaged procedures and functions	None

5

Creating Data Dictionary Views and Standard Packages

Objectives

Objectives

After completing this lesson, you should be able to do the following:

- **Construct the data dictionary views**
- **Query the data dictionary**
- **Prepare the PL/SQL environment using the administrative scripts**
- **Administer stored procedures and packages**
- **List the types of database event triggers**

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

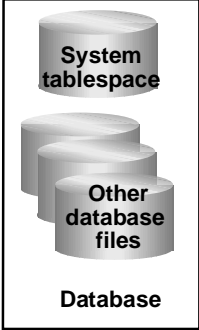
Overview

Overview

Other objects created with the database:

- **Data dictionary**
 - **Base tables**
 - **Views**
- **Dynamic performance tables**
- **Built-in PL/SQL packages**

Database event triggers fire automatically during a specific event, such as a server error.



The diagram shows a stack of three cylinders. The top cylinder is labeled 'System tablespace'. The middle and bottom cylinders are grouped together and labeled 'Other database files'. Below the cylinders, the word 'Database' is written.

ORACLE®

Copyright © Oracle Corporation, 1999. All rights reserved.

Built-In Database Objects

In addition to creating the database files, the Oracle server creates structures within the data files.

- The data dictionary contains descriptions of the objects in the database. It includes two types of objects:
 - *Base tables* are the underlying tables that store the description of the associated database.
 - *Data dictionary views* summarize and display the information stored in the base tables.
- *Dynamic performance tables* contain information used by the database administrator (DBA) to monitor and tune the database and instance.
- Built-in PL/SQL program units add functionality to the database.

Database Event Triggers

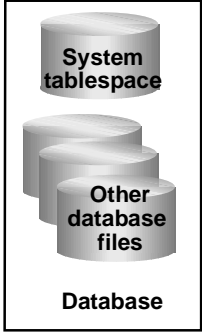
Triggers are procedures that execute (“fire”) implicitly whenever a table or view is modified, or when some user actions or database system actions occur.

This lesson covers the creation and use of data dictionary views, PL/SQL packages, and database event triggers.

Data Dictionary Overview

Data Dictionary

- **Central to every Oracle database**
- **Describes the database and its objects**
- **Contains read-only tables and views**
- **Updated by SQL commands:**
 - DDL
 - Some DML
- **Owned by the user SYS**
- **Stored in the SYSTEM tablespace**
- **Accessed with SELECT statements**



The diagram shows a stack of three cylinders representing database components. The top cylinder is labeled 'System tablespace'. The middle and bottom cylinders are grouped together and labeled 'Other database files'. The entire stack is labeled 'Database' at the bottom.

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE®

Data Dictionary

One of the most important parts of an Oracle database is its data dictionary, which is a read-only set of tables and views that provides information about its associated database.

The data dictionary is updated by the Oracle server whenever a data definition language (DDL) command is executed. In addition, data manipulation language (DML) commands, such as one that causes a table to extend, can update the data dictionary.

Not only is the data dictionary central to every Oracle database, it is an important source of information for all users, from end users to application designers and database administrators. To access the data dictionary, you use SQL statements. Because the data dictionary is read-only, you can issue only queries against the tables and views of the data dictionary.

Data Dictionary Contents

Data Dictionary Contents

The data dictionary provides information about:

- Logical and physical database structure
- Definitions and space allocations of objects
- Integrity constraints
- Users
- Roles
- Privileges
- Auditing
- Other information

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Data Dictionary Contents

A data dictionary contains:

- The definitions of all schema objects in the database (tables, views, indexes, clusters, synonyms, sequences, procedures, functions, packages, triggers, and so on)
- How much space has been allocated for, and is currently used by, the schema objects
- Default values for columns
- Integrity constraint information
- The names of Oracle users
- Privileges and roles each user has been granted
- Auditing information, such as who has accessed or updated various schema objects
- Other general database information

Base Tables and Data Dictionary Views

Base Tables and Data Dictionary Views

The data dictionary contains two parts:

- **Base tables:**
 - Normalized
 - Created with database using `sql.bsq` script
- **Data dictionary views:**
 - Used to simplify the base table information
 - Accessed through public synonyms
 - Created with the `catalog.sql` script

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Base Tables

Base tables are the underlying tables that store information about the associated database. The data dictionary base tables are the first objects created in any Oracle database. They are automatically created when the Oracle server runs the `sql.bsq` script as the database is created. Only the Oracle server should write to these tables. Users rarely access them directly because most of the data is stored in a cryptic format. Never use DML commands to update the base data dictionary tables directly, with the exception of the AUD\$ table. The AUD\$ table is discussed in the lesson “Managing Privileges.”

An example of a base table is the IND\$ table, which contains information about the indexes in the database.

Data Dictionary Views

Most users examine the data dictionary by selecting from the views rather than the base tables. These views summarize and display the information stored in the base tables. They decode the base table data into useful information, using joins and WHERE clauses to simplify the information. For example, in the data dictionary views, object names are used, instead of only object numbers.

How the Data Dictionary Is Used

How the Data Dictionary Is Used

The data dictionary has three primary uses:

- The Oracle server uses it to find information about:
 - Users
 - Schema objects
 - Storage structures
- The Oracle server modifies it when a DDL statement is executed.
- Users and DBAs can use it as a read-only reference for information about the database.

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

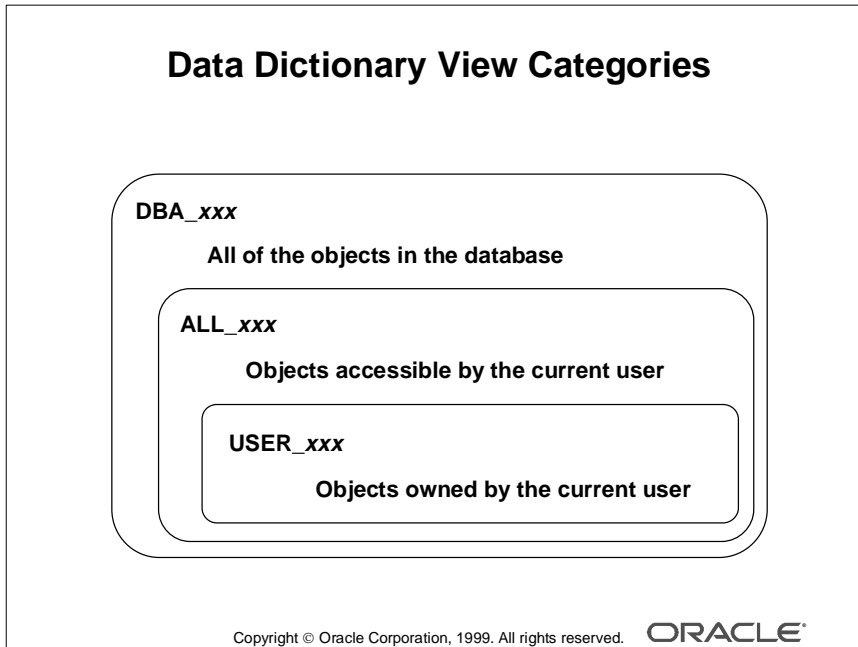
How the Oracle Server Uses the Data Dictionary

Data in the base tables of the data dictionary is necessary for the Oracle server to function. Therefore, only the Oracle server should write or change data dictionary information. During database operation, the Oracle server reads the data dictionary to ascertain that schema objects exist and that users have proper access to them. The Oracle server also updates the data dictionary continuously to reflect changes in database structures, auditing, grants, and data structures.

How Users and Database Administrators Can Use the Data Dictionary

The views of the data dictionary serve as a reference for all database users. You access the data dictionary views by using the SQL language. Some views are accessible to all Oracle users; others are intended for database administrators only.

Data Dictionary View Categories



Three Data Dictionary View Categories

Data dictionary views are split into three categories, distinguishable from each other by their prefixes:

- **DBA:** Database administrator's view; that is, what is in all schemas
- **ALL:** Expanded user's view; that is, what the user can access
- **USER:** User's view; that is, what is in the user's schema

All of the views have public synonyms created on them.

Not all data dictionary views use this naming convention.

Views with the DBA Prefix

Views with the DBA prefix show a global view of the entire database. They are meant to be queried only by database administrators. Any user granted the system privilege **SELECT ANY TABLE** can query the DBA-prefixed views of the data dictionary.

To query on all objects in the database, the DBA could issue the following statement:

```
SELECT  owner, object_name, object_type
FROM    dba_objects;
```

Views with the ALL Prefix

Views with the ALL prefix refer to the user's overall perspective of the database. These views return information about schema objects to which the user has access by way of public or explicit grants of privileges and roles, in addition to schema objects that the user owns.

For example, the following query returns information about all the objects to which you have access:

```
SELECT    owner, object_name, object_type
FROM      all_objects;
```

Views with the USER Prefix

The views most likely to be of interest to typical database users are those with the USER prefix. These views:

- Refer to the user's own private environment in the database
- Generally refer to objects owned by the current user
- Have columns identical to the other views, except that the column OWNER is implied to be the current user
- Return a subset of the information in the ALL_ views
- Can have abbreviated PUBLIC synonyms for convenience

For example, the following query returns all the objects contained in your schema:

```
SELECT    object_name, object_type
FROM      user_objects;
```

Data Dictionary Examples

Data Dictionary Examples

- General overview
 - DICT_COLUMNS
- Schema objects
 - DBA_TABLES
 - DBA_TAB_COLUMNS
 - DBA_OBJECTS
 - DBA_CONSTRAINTS
- Space allocation
 - DBA_SEGMENTS
 - DBA_EXTENTS
 - DBA_FREE_SPACE
- Database structure
 - DBA_DATA_FILES
 - DBA_ROLLBACK_SEGS
 - DBA_TABLESPACES

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE®

Data Dictionary Examples

To get an overview of the data dictionary and dynamic performance views, you can query the DICT_COLUMNS view or its synonym DICT.

```
SQL>SELECT *
2 FROM dictionary
3 WHERE UPPER(comments) LIKE '%TABLE%';
```

TABLE_NAME	COMMENTS
ALL_TABLES	Description of relational tables accessible to the user
ALL_UPDATABLE_COLUMNS	Description of updatable columns
...	

To get an overview of the columns in the data dictionary and dynamic performance views, you can query the DICT_COLUMNS view.

Note: See the *Oracle8i Reference* for a complete list of data dictionary views and their columns.

Dynamic Performance Views

Dynamic Performance Tables

- “Virtual” tables
- Information accessed from:
 - Memory
 - Control file
- Synonyms begin with V\$
- DBA uses to monitor and tune the database
- Listed in V\$FIXED_TABLE
- Example: V\$DATAFILE for data file information

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Dynamic Performance Views

Throughout its operation, the Oracle instance records current database activity in a set of “virtual” tables called dynamic performance views.

They are not true tables, and are not to be accessed by most users; however, database administrators can query, grant the SELECT privilege, and create views on these views. These views are sometimes called fixed views because they cannot be altered or removed by the database administrator.

SYS owns the dynamic performance tables; their names all begin with V\$. Views are created on these tables. The view names all begin with V_\$. Then public synonyms are created for the views. The synonym names also begin with V\$.

Note: See the *Oracle8i Reference* for a complete list of the dynamic performance views and their columns.

Stored Program Units

Stored Program Units

- **PL/SQL**
 - Oracle's procedural language extension to SQL
 - Stored in the data dictionary
- **Java**
 - Stored in the data dictionary
 - To execute, publish its call specification
- **External procedures**
 - Written in C
 - Stored in a shared library
 - To execute, publish the call specification

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Stored Program Units

The Oracle server enables you to access and manipulate database information using procedural schema objects called stored program units. Stored program units are a logically related set of SQL and programming language statements that perform a specific task. They can be called from both SQL and PL/SQL.

PL/SQL Program Units

PL/SQL is Oracle's procedural language extension to SQL. PL/SQL enables you to mix SQL statements with procedural constructs.

Java Program Units

The database administrator can install the JServer component of the Oracle server to execute Java programs units. To call a Java method from SQL or PL/SQL, you publish the method by writing a call specification. The call specification maps the Java method names, parameter types, and return types to their SQL counterparts.

C Program Units

An external procedure is a C program that is stored in a shared library and can be called from a PL/SQL program. External procedures enable the database administrator to extend the functionality of the Oracle server. They execute in a separate address space from that of the Oracle server. To call an external procedure, its name, parameter types, and return type must be published.

Benefits

Stored program units provide the following advantages:

- To reduce compile times, the PL/SQL code is precompiled and stored in the data dictionary with the source code.
- Java and C programs can be called from SQL and PL/SQL by defining their call specification to PL/SQL.
- They are stored in the shared pool to reduce disk retrieval.
- Data security can be enforced by letting users access data only through procedures and functions.
- During execution, multiple users share a single copy of the program unit.
- Stored functions can be used in SQL expressions, in the same manner as built-in Oracle functions, such as UPPER and SUBSTR.

Note: This section presents an overview of the stored procedures to enable you to administer the stored units. Developing and maintaining stored procedures, packages, and triggers are covered in detail in the course *PL/SQL Program Units*.

Stored PL/SQL Program Units

Stored PL/SQL Program Units

- A procedure, function, package, or trigger
- Can accept and return parameters
- Functions can be called from SQL
- A set of SQL and PL/SQL commands
- Stored in the data dictionary
- Loaded into the shared pool

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Stored PL/SQL Program Units

Stored PL/SQL program units are:

- A logically related set of SQL and other PL/SQL programming language statements that perform a specific task
- Procedures, functions, triggers, or packages
- Assigned a name that is used when calling the program unit
- Created or removed by developers with a CREATE or DROP command.
- Executed from an Oracle application or by using Oracle tools such as SQL*Plus or Enterprise Manager
- Created and stored in the data dictionary as a schema object
- Loaded into the shared pool

Stored Procedures

A stored procedure is a procedure or function that is created and stored in the data dictionary as a schema object. It consists of a set of PL/SQL and SQL constructs.

Procedures and functions are identical except that functions always return a single value to the caller, while procedures do not. For simplicity, the term “procedure” as used in the remainder of this lesson means “procedure or function.”

Triggers

A trigger is a PL/SQL program unit that is executed implicitly by the Oracle server when a specific type of event occurs. The trigger is never called; it only executes when the event occurs. Triggers are covered in more detail in a subsequent section.

Parameters

Procedures and functions provide parameters that can be input only (IN), output only (OUT), or both input and output parameters (IN OUT). The IN mode is the default.

Packages

Stored PL/SQL Packages

A package:

- **Groups logically related PL/SQL types, items, and subprograms**
- **Has two parts:**
 - **A specification describes its components**
 - **A body implements the logic**
- **Example: DBMS_SESSION.SET_ROLE**
 - **Package: DBMS_SESSION**
 - **Procedure: SET_ROLE**

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Packages

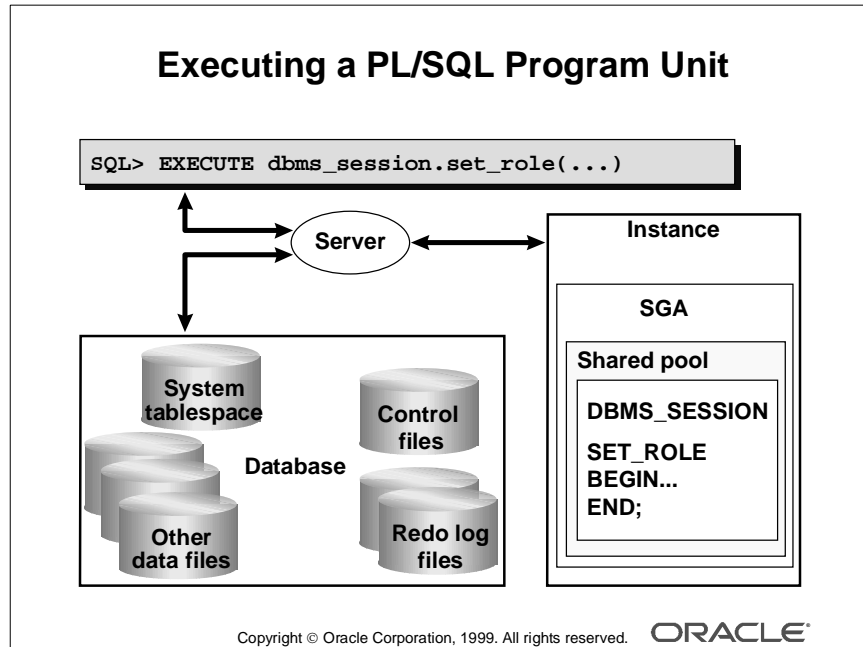
A package is a group of functionally related variables, constants, cursors, exceptions, procedures, and functions stored together in the database for use as a unit. Similar to stand-alone procedures and functions, packaged procedures and functions can be called explicitly by applications or users.

A package usually has two parts stored separately in the database:

- The specification is the interface to the application and declares the types, variables, constants, exceptions, cursors, and subprograms available for use outside of the package.
- The body implements the specification. It includes the PL/SQL code to implement the procedure and function specifications included in the package specification. It may also include procedures and functions that are callable only from inside of the package.

The functionality of a package is similar to that of stored procedures. Once written and compiled, the contents can be shared by many applications. One major benefit is that the first time a package construct is called, the whole package is loaded into memory.

Executing a PL/SQL Program Unit



Calling a PL/SQL Program Unit

In the slide, the `DBMS_SESSION.SET_ROLE` procedure is executed using SQL*Plus. The procedure is explained in a subsequent section.

The `EXECUTE` command is a SQL*Plus command that calls a stored procedure.

The package name is `DBMS_SESSION` and the `SET_ROLE` procedure is defined within the package. The procedure is called with the syntax:

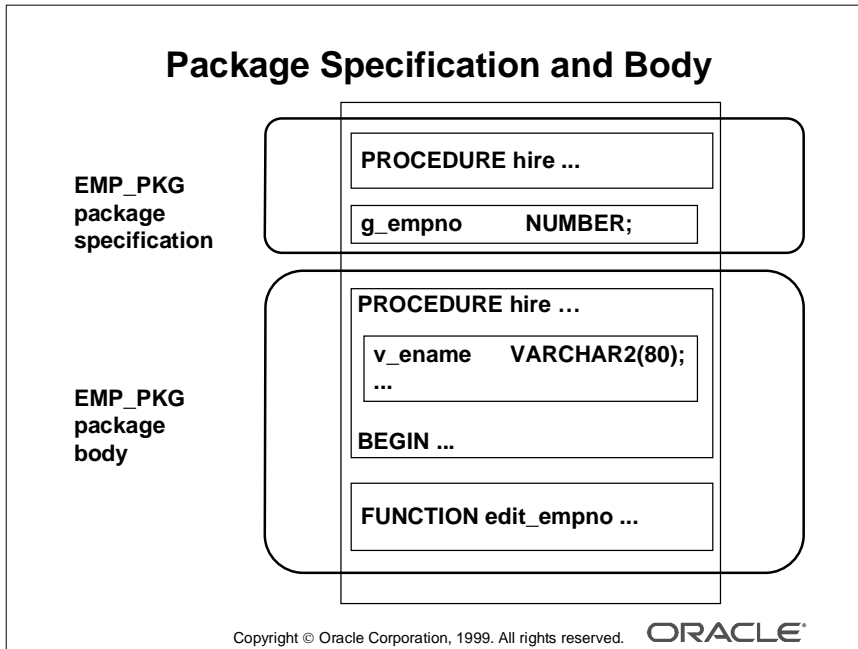
```
package_name.procedure_name
```

Parameters are coded within the parentheses.

Executing a PL/SQL Program Unit

The server process reads the stored PL/SQL program unit into the shared pool and executes it. When a package is referenced, the entire package is read into the shared pool.

Package Specification and Body



EMP_PKG Package Definition

The slide shows the EMP_PKG package.

- The HIRE procedure is callable from outside of the package, because it is declared in the package specification. The definition of the procedure, that is, its PL/SQL and SQL code, is included in the definition of the procedure, which is in the package body. To execute the HIRE procedure from SQL*Plus, use the EXECUTE command.

```
SQL> EXECUTE emp_pkg.hire(100, 'Moirra')
PL/SQL procedure successfully completed.
```

- The G_EMPNO global variable can be referenced from within or outside of the package. You can use the EXECUTE command in SQL*Plus to assign a value of 10 to the global variables.

```
SQL> EXEC emp_pkg.g_empno := 10;
PL/SQL procedure successfully completed.
```

- The V_NAME local variable can be referenced only from within HIRE.
- The EDIT_EMPNO function can be called only from within the package, because it is not declared in the package specification.

Oracle-Supplied Packages

Oracle-Supplied Packages

- **DBMS_SESSION:** Generates SQL commands such as ALTER SESSION or SET ROLE
- **DBMS_UTILITY:** Provides various utility routines
- **DBMS_SPACE:** Provides segment space availability information
- **DBMS_ROWID:** Provides ROWID information
- **DBMS_SHARED_POOL:** Keeps and unkeeps packages in the shared pool
- **DBMS_LOB:** Provides routines for operations on BLOB and CLOB data types

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Oracle-Supplied Packages

The slide lists some of the Oracle-supplied, or built-in, packages.

The table lists examples of procedures in the Oracle-supplied packages.

Package	Package Procedures
DBMS_SESSION	SET_ROLE SET_SQL_TRACE
DBMS_UTILITY	ANALYZE_SCHEMA COMPILE_SCHEMA DB_VERSION
DBMS_ROWID	ROWID_INFO
DBMS_SPACE	UNUSED_SPACE FREE_BLOCKS
DBMS_SHARED_POOL	KEEP UNKEEP

Note: For more information on Oracle-supplied packages, see *Oracle8i Supplied Packages Reference* and *Oracle8i PL/SQL User's Guide and Reference*.

Obtaining Information

Obtaining Information About Stored Objects

- Data dictionary view **DBA_OBJECTS**:
 - **OWNER**
 - **OBJECT_NAME**
 - **OBJECT_TYPE**
 - **STATUS**:
 - **VALID**
 - **INVALID**
- **DESCRIBE** command:

```
SQL> DESCRIBE dbms_session
```

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Dependencies

The Oracle server automatically records dependencies among objects in the data dictionary. For example, a procedure can be dependent on a view, and a view again is dependent on a table. If the table on which the view is built is dropped, the Oracle server sets the **STATUS** column of the data dictionary view **DBA_OBJECTS** for the dependent view and procedure to **INVALID**.

All schema objects in a database have a status of **VALID** or **INVALID**.

- **VALID**: The object has been compiled and can be immediately used when referenced.
- **INVALID**: The object must be compiled before it can be used. In the case of procedures, functions, and packages, this means compiling the object. In the case of views, this means that the view must be reparsed. If these objects are still invalid after recompiling, then they may require code modifications before they can be executed.

Dependencies (continued)

Query the data dictionary view DBA_OBJECTS to obtain information about the owner, name, type, and status of the objects in the database.

```
SQL> SELECT object_name, object_type, status
       2 FROM dba_objects WHERE object_name like 'DBMS_%'
```

OBJECT_NAME	OBJECT_TYPE	STATUS
DBMS_ALERT	PACKAGE	VALID
DBMS_ALERT	PACKAGE BODY	VALID
DBMS_ALERT_INFO	TABLE	VALID
DBMS_APPLICATION_INF	PACKAGE	VALID
DBMS_APPLICATION_INF	PACKAGE BODY	VALID
DBMS_AQ	PACKAGE	VALID
DBMS_AQ	PACKAGE BODY	VALID

...

Run the DESCRIBE command in SQL*Plus for specifications for all of the procedures and functions of a package.

```
SQL> desc dbms_session
```

...

```
PROCEDURE SET-NLS
```

Argument	Name	Type	In/Out	Default?
----------	------	------	--------	----------

PARAM		VARCHAR2	IN	
-------	--	----------	----	--

VALUE		VARCHAR2	IN	
-------	--	----------	----	--

```
PROCEDURE SET_ROLE
```

Argument	Name	Type	In/Out	Default?
----------	------	------	--------	----------

ROLE_CMD		VARCHAR2	IN	
----------	--	----------	----	--

```
PROCEDURE SET_SQL_TRACE
```

Argument	Name	Type	In/Out	Default?
----------	------	------	--------	----------

SQL_TRACE		BOOLEAN	IN	
-----------	--	---------	----	--

...

Troubleshooting

Troubleshooting

- The status of dependent objects may be **INVALID**:
 - When DDL commands are executed on referenced objects
 - After creating the objects using the **IMPORT** utility
 - When a dependent object is modified
- For program units with an **INVALID** status:
 - The Oracle server automatically recompiles
 - The user can recompile manually

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Troubleshooting

After the execution of DDL commands such as **ALTER TABLE ADD**, **RENAME**, **DROP**, and **CREATE OR REPLACE**, the status of the dependent objects changes to **INVALID**.

Also, loading dependent views and stored procedures with the Import utility may lead to an **INVALID** object status, because the Import utility may not be able to create the dependent objects after creating the referenced objects.

Object A is dependent on object B if A references B. For example, the **EMP** table, which is referenced in the **HIRE_EMP** procedure, has a column added to it. The **HIRE_EMP** procedure is marked as **INVALID** and must be recompiled before it can be executed.

The Oracle server automatically recompiles an invalid view or PL/SQL program unit the next time it is used. In addition, the user can force the Oracle server to recompile a view, stored procedure, or package by using the appropriate SQL command.

Note: Recompiling stored procedures and packages is covered in detail in the course *PL/SQL Program Units*.

Constructing the Data Dictionary

Creating Data Dictionary Views

Script	Purpose
<code>catalog.sql</code>	Creates commonly used data dictionary views and synonyms
<code>catproc.sql</code>	Runs scripts required for server-side PL/SQL

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Creating the Data Dictionary

The base tables of the data dictionary are automatically created when a database is created. When the database is created with the Oracle Universal Installer, the data dictionary views and scripts for your Oracle server options are run automatically.

You might need to run them again when upgrading to a new release of the Oracle server. Run these scripts connected to the Oracle server as the user SYS. They are located in:

- `$ORACLE_HOME/rdbms/admin` directory on UNIX
- `%ORACLE_HOME%\rdbms\admin` directory on NT

The other administrative scripts are stored in these directories.

The `catalog.sql` Script

The `catalog.sql` script creates the views on the base tables and on the dynamic performance views, and their synonyms. It starts other scripts that create objects for:

- The basic PL/SQL environment, including declarations for PL/SQL:
 - Data types
 - Predefined exceptions
 - Built-in procedures and functions
 - SQL operations
- Auditing
- Import/export
- SQL*Loader
- Installed options

The `catproc.sql` Script

The `catproc.sql` script establishes the usage of the PL/SQL functionality. In addition, it creates several of the PL/SQL packages that are used to extend the RDBMS functionality. The `catproc.sql` script also creates additional packages and views for:

- Alerts
- Pipes
- Logminer
- Large objects
- Objects
- Advanced queuing
- Replication option
- Other built-ins and options

Note: You may use additional scripts to support other extended Oracle server functionality, as discussed in the next section.

- For more information about built-in packages, see the *Oracle8i Supplied Packages Reference*.
- For more information about scripts, see the *Oracle8i Administrator's Guide*.

Administrative Scripts

Administrative Scripts: Naming Conventions

Convention	Description
cat*.sql	Catalog and data dictionary information
dbms*.sql	Database package specifications
prvt*.plb	Wrapped database package code
utl*.sql	Views and tables for database utilities

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Administrative Script Categories

The administrative scripts can be separated into four categories by their filenames:

- cat*.sql
- dbms*.sql
- prvt*.plb
- utl*.sql

The cat*.sql Scripts The cat*.sql scripts create data dictionary views. In addition to the catalog.sql and catproc.sql scripts, there are scripts that create information for Oracle utilities.

For example, the catadt.sql script creates data dictionary views for showing metadata information for types and other object features in the RDBMS. The catnoadt.sql script drops these tables and views.

Administrative Script Categories (continued)

The `dbms*.sql` and `prvt*.plb` Scripts The `dbms*.sql` and `prvt*.plb` scripts create objects for predefined Oracle packages that extend the Oracle server functionality. These programs simplify the task of administering the database.

Most SQL scripts are run during the execution of the `catproc.sql` script. A few additional scripts must be executed by the database administrator.

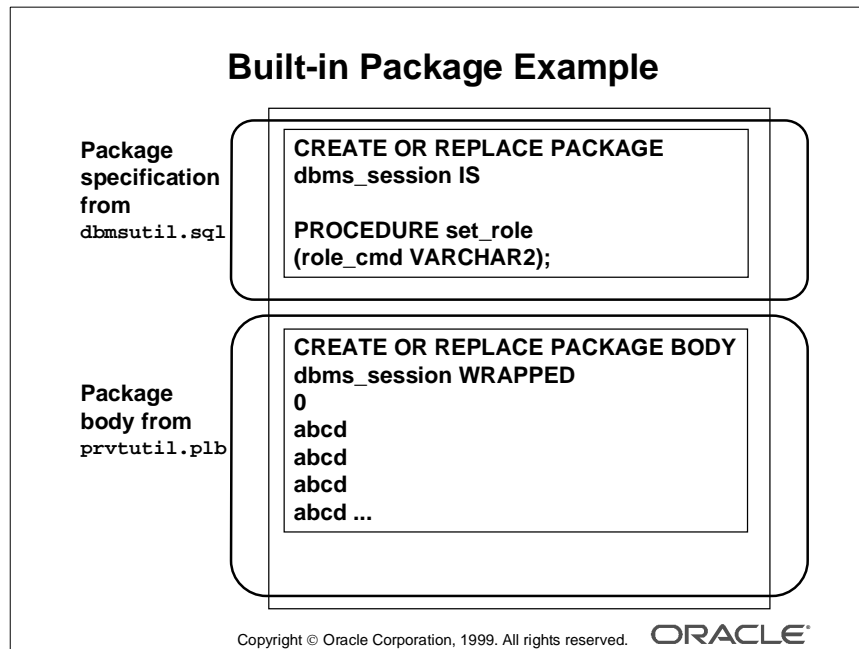
An example is the `dbmspool.sql` script, which enables you to display the sizes of objects in the shared pool and mark them to be kept or removed in the SGA in order to reduce shared pool fragmentation.

The `utl*.sql` Scripts The `utl*.sql` scripts must be run when the database needs additional views and tables.

For example, the script `utlxplan.sql` creates a table used to view the execution plan of a SQL statement.

Note: Most of these scripts must be executed under the user `sys`. The database administrator should examine the scripts to find out which user account must be used to run the scripts.

Built-in Package Example



DBMS_SESSION Example

The DBMS_SESSION package specification is located in the `dbmsutil.sql` script, and its package body is located in the `prvtutil.plb` script. The `.plb` extension is used to indicate PL/SQL binaries.

The package body has been created with the Oracle Wrapper utility, which hides application internals by encrypting PL/SQL source code.

To execute the SET_ROLE package procedure with the required IN parameter value, the name of the role, in SQL*Plus, use the following statement:

```
SQL> execute DBMS_SESSION.SET_ROLE( 'APP1' );
Statement processed.
```

This generates the SET ROLE command, appending the text APP1 to SET ROLE and then executing it as an SQL command. This command enables or disables roles and will be covered in detail in the lesson “Managing Roles.”

Because the two scripts, `dbmsutil.sql` and `prvtutil.plb`, are called from `catproc.sql`, they do not need to be run by the database administrator after installation.

Packages and Scripts

The packages listed in the earlier slide are created from their own script files.

- DBMS_SPACE, DBMS_UTILITY, DBMS_ROWID, and DBMS_SESSION are defined by the scripts `dbmsutil.sql` and `prvtutil.plb`. These scripts are called from `catproc.sql`.
- DBMS_LOB is defined by `dbmslob.sql` and `prvtlob.plb`.
- DBMS_SHARED_POOL is created by running the `dbmspool.sql` script.

Triggers

Triggers

- **Written in PL/SQL, Java, or C**
- **Fire:**
 - **Automatically**
 - **Implicitly**
- **Stored in the database**
- **Managed like other stored program units**
- **Usage examples:**
 - **Generate derived column values**
 - **Enforce complex edits**
 - **Log events**
 - **Audit**
 - **Gather statistics**

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Overview

Triggers are procedures written in PL/SQL, Java, or C that execute (“fire”) implicitly when a table is modified, an object is created, or some user actions or database system actions occur. These procedures can be written in PL/SQL or Java and stored in the database, or they can be written as external procedures.

Triggers are similar to stored procedures; however, procedures and triggers differ in the way that they are invoked. A procedure is explicitly executed by a user, application, or trigger. Triggers (one or more) are implicitly fired (executed) by the Oracle server when a triggering event occurs, no matter which user is connected or which application is being used.

How Triggers Are Used

Triggers can supplement the standard capabilities of the Oracle server to provide a highly customized database management system. For example, triggers can be used to:

- Automatically generate derived column values
- Prevent invalid transactions
- Enforce complex security authorizations
- Enforce referential integrity across nodes in a distributed database
- Enforce complex business rules that cannot be coded in constraints
- Provide transparent event logging
- Provide sophisticated auditing
- Maintain synchronous table replicates
- Gather statistics on table access
- Modify table data when DML statements are issued against views
- Publish information about database events, user events, and SQL statements to subscribing applications

Parts of a Trigger

Parts of a Trigger

- **Triggering event**
 - Instance startup or shutdown
 - A specific error message or any error message
 - User logon or logoff
 - A DML statement on a specific table or view
 - A DDL statement on any schema
- **Restriction**
 - Optional boolean clause in a WHEN statement
 - Trigger only fires if clause is TRUE
- **Action: Program unit block**

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Three Parts to a Trigger

A trigger has three basic parts:

- A triggering event or statement
- A trigger restriction
- A trigger action

Triggering Event or Statement

A triggering event or statement is the SQL statement, database event, or user event that causes a trigger to be fired. The following triggering events are most likely to be used by a database administrator:

- An instance startup or shutdown
- A specific error message or any error message
- A user logon or logoff
- A CREATE, ALTER, or DROP statement on any schema

In addition, application developers may write triggers that fire when an INSERT, UPDATE, or DELETE statement is executed on a specific table or view.

Trigger Restriction

A trigger restriction specifies a Boolean (logical) expression that must be TRUE for the trigger to fire. The trigger action is not executed if the trigger restriction evaluates to FALSE or UNKNOWN. The trigger restriction is coded with a WHEN clause.

Trigger Action

A trigger action is the procedure (PL/SQL block, Java program, or C callout) that contains the SQL statements and code to be executed when a triggering statement is issued and the trigger restriction evaluates to TRUE. Like stored procedures, a trigger action can contain SQL and PL/SQL or Java statements, define PL/SQL language constructs (variables, constants, cursors, exceptions, and so on) or Java language constructs, and call stored procedures.

Trigger Example

Trigger Example

```
CREATE OR REPLACE TRIGGER log_logon
  AFTER LOGON ON DATABASE
  WHEN (USER = 'SYS' OR USER LIKE 'OPS$%')
BEGIN
  INSERT INTO sys.event_log
    VALUES ('Logon ' || USER || ' at '
      || TO_CHAR(sysdate, 'YYYY-MM-DD HH24:MI:SS'));
  COMMIT;
END;
/
```

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Logon Trigger

The example in the slide shows a trigger being created or replaced with the CREATE OR REPLACE TRIGGER command. The trigger has the following characteristics:

- It is named LOG_LOGON.
- It fires after any user logs on to the instance.
- It will only execute if the username is SYS or the username is prefixed with OPS\$.
- When it executes, it inserts a row into the EVENT_LOG table to show that the user logged on to the instance.

Trigger information is stored in the data dictionary view DBA_TRIGGERS.

The implementation of triggers is covered in more detail in the course *PL/SQL Stored Program Units*.

Summary

Summary

In this lesson, you should have learned how to:

- Use the data dictionary views to get information about the database and instance
- Obtain information about data dictionary views from **DICTIONARY** and **DICT_COLUMNS**
- Use the administrative scripts to create data dictionary views and Oracle-supplied packages
- Obtain information about stored PL/SQL from **DBA_SOURCE**
- Use database event triggers

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Quick Reference

Context	Reference
Initialization parameters	None
Dynamic performance views	None
Data dictionary views	DICTIONARY DICT DICT_COLUMNS DBA_OBJECTS DBA_TRIGGERS
Commands	None
Packaged procedures and functions	DBMS_SESSION.SET_ROLE

6

Maintaining the Control File

Objectives

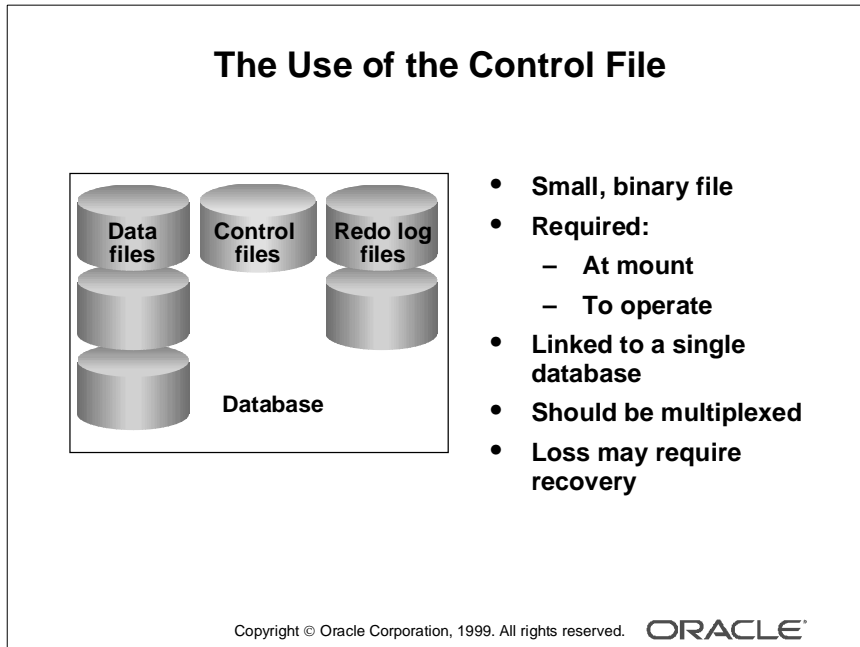
Objectives

After completing this lesson, you should be able to do the following:

- **Explain the uses of the control file**
- **List the contents of the control file**
- **Multiplex the control file**
- **Obtain control file information**

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

The Use of the Control File



The Use of the Control File

The control file of a database is a small binary file necessary for the database to start and operate successfully. Each control file is associated with only one Oracle database. A control file is updated continuously by the Oracle server during database use, so it must be available for writing whenever the database is open. The information in the control file can be modified only by the Oracle server; no DBA or end user can edit the control file.

If for some reason the control file is not accessible, the database will not function properly. If all copies of a database's control files are lost, the database must be recovered before it can be opened.

Control File Contents

Control File Contents

- Database name and identifier
- Database creation date
- Data file and redo log locations
- Tablespace names
- Log history
- Backup information
- Current log sequence number
- Checkpoint information

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE

Contents of the Control File

- The database name is taken from either the name specified by the initialization parameter DB_NAME or the name used in the CREATE DATABASE statement.
- The database identifier is recorded when the database is created.
- The time stamp of the database creation is also recorded when the database is created.
- The names and locations of associated data files and online redo log files are updated when a data file or redo log is added to, renamed in, or dropped from the database.
- Tablespace information is updated as tablespaces are added or dropped.
- The log history is recorded during log switches.
- The location and status of archived logs are recorded when archiving occurs.
- The location and status of backups are recorded by the Recovery Manager utility.
- The current log sequence number is recorded when log switches occur.
- Checkpoint information is recorded as checkpoints are made.

Contents of the Control File (continued)

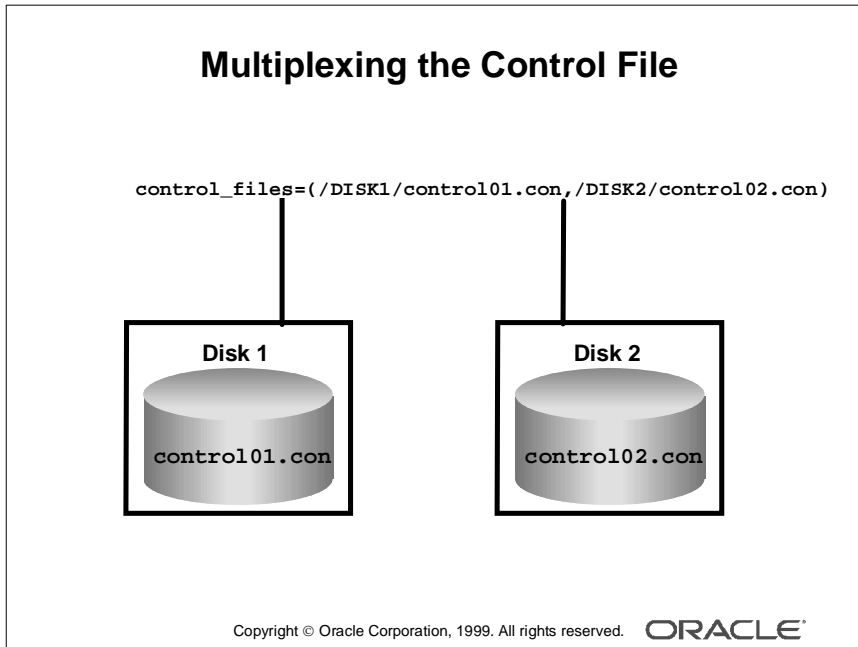
The control file consists of two types of sections:

- Reusable
- Not reusable

Reusable sections store Recovery Manager information, such as backup data file names and backup redo log file names. They are used in a circular manner and can be reused only by Recovery Manager.

Note: Recovery Manager is covered in more detail in the course *Enterprise DBA Part 1B: Backup and Recovery*.

Multiplexing the Control File



Multiplexing the Control File

As with online redo log files, Oracle allows multiple, identical control files to be open concurrently and written to. You can specify up to eight fully qualified control file names, using the initialization parameter `CONTROL_FILES`. The Oracle server creates and maintains all files listed in this parameter when the instance is started.

You can multiplex control files by:

- Creating multiple control files when the database is created
- Adding a control file after the database is created

Creating a Control File with the Database The easiest method for creating multiple control files is to include the control file names in the `CONTROL_FILES` initialization parameter when the database is created:

```
CONTROL_FILES = (/DISK1/control01.con, /DISK2/control02.con)
```

The Oracle server will create all of the control files listed in the parameter. The filenames specified in this parameter should include the full pathname. Filename specification is operating system-specific.

How to Add a Control File To add a control file or change the number or location of the control file:

- 1 Shut down the database.
- 2 Copy the current control file using an operating system command.
- 3 Add the new control file names to the `CONTROL_FILES` parameter.
- 4 Start up the database.

Guidelines for Control Files

Guidelines for Control Files

- **You should:**
 - **Multiplex the control file**
 - **Include the full pathname in CONTROL_FILES**
 - **Back up the control file after the database structure changes**
- **Control files:**
 - **Are sized by CREATE DATABASE keywords**
 - **Have a reusable section that can expand due to Recovery Manager updates**

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

Purpose of Multiplexing

To safeguard against a single point of failure of the control file, it is strongly recommended that you multiplex the control file, storing each copy on a different physical disk.

If a control file is lost, use the copy of the control file to restart the instance.

If you have multiple copies of the current control file on different disks, you can easily start an instance without the need for database recovery.

CREATE DATABASE Keywords

Keywords specified during the creation of the database affect the size of the control file. This is particularly significant when the parameters have large values. You might need to re-create the control file to change one or more of the database limit parameters, which could increase or decrease the size of the control file.

The size of the control file is influenced by the following keywords in the CREATE DATABASE or CREATE CONTROLFILE commands:

- MAXLOGFILES
- MAXLOGMEMBERS
- MAXLOGHISTORY
- MAXDATAFILES
- MAXINSTANCES

A new control file must be created to change the size of the space created by these keywords.

Reusable Section Can Expand

When Recovery Manager is used, the reusable section of the control file can expand based on the number of entries required by Recovery Manager.

Backup After Database Structure Changes

Because the control file records the physical structure of the database, you should immediately make a backup of your control file after making changes to the physical structure of the database.

Backup and recovery of the control file is covered in detail in the course *Enterprise DBA Part 1B: Backup and Recovery*.

Obtaining Information About the Control File

Obtaining Information

- **V\$CONTROLFILE**
- **V\$CONTROLFILE_RECORD_SECTION**
- **Performance views from the control file:**
 - **V\$DATAFILE**
 - **V\$TEMPFILE**
 - **V\$TABLESPACE**
 - **V\$LOG**
 - **Others**

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

Obtaining Information About the Control File

To obtain the location and names of the control files, use the dynamic performance view **V\$CONTROLFILE**.

```
SQL> SELECT name
      2  FROM v$controlfile;
NAME
-----
/DISK1/control01.con
/DISK2/control02.con
2 rows selected.
```

The **V\$PARAMETER** view can also be used; however, because of column lengths, the control file names may be truncated.

Obtaining Information About the Control File (continued)

To obtain information about the different sections of the control files, query the V\$CONTROLFILE_RECORD_SECTION dynamic performance view.

```
SQL> SELECT type, record_size, records_total, records_used
2 FROM v$controlfile_record_section
3 WHERE type='DATAFILE';
```

TYPE	RECORD_SIZ	RECORDS_TO	RECORDS_US
-----	-----	-----	-----
DATAFILE	180	30	4

1 row selected.

The column RECORDS_TO specifies the number of records allocated for a special section. For example, you can view the maximum number of data files—30, in our example, which is determined by the MAXDATAFILES parameter in the CREATE DATABASE command.

The information in several of the other dynamic performance views is obtained from the control file:

- V\$BACKUP
- V\$DATAFILE
- V\$TEMPFILE
- V\$TABLESPACE
- V\$ARCHIVE
- V\$LOG
- V\$LOGFILE
- V\$LOGHIST
- V\$ARCHIVED_LOG
- V\$DATABASE
- Others

Summary

Summary

In this lesson, you should have learned how to:

- Explain that the control file is required to mount and operate the database
- Multiplex the control file to avoid a single point of database failure
- Back up the control file after making changes to the physical database structure

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

Quick Reference

Context	Reference
Initialization parameters	CONTROL_FILES
Dynamic performance views	V\$CONTROLFILE V\$CONTROLFILE_RECORD_SECTION V\$PARAMETER
Data dictionary views	None
Commands	None
Packaged procedures and functions	None

7

Maintaining Redo Log Files

Objectives

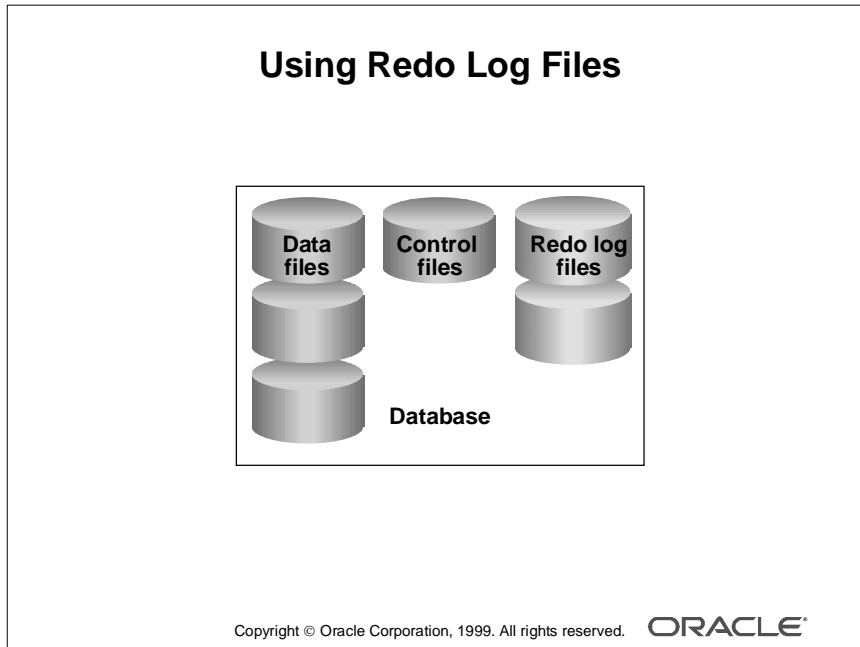
Objectives

After completing this lesson, you should be able to do the following:

- **Explain the use of online redo log files**
- **Obtain log and archive information**
- **Control log switches and checkpoints**
- **Multiplex and maintain online redo log files**
- **Plan online redo log files**
- **Troubleshoot common redo log file problems**
- **Analyze online and archived redo logs**

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

Overview

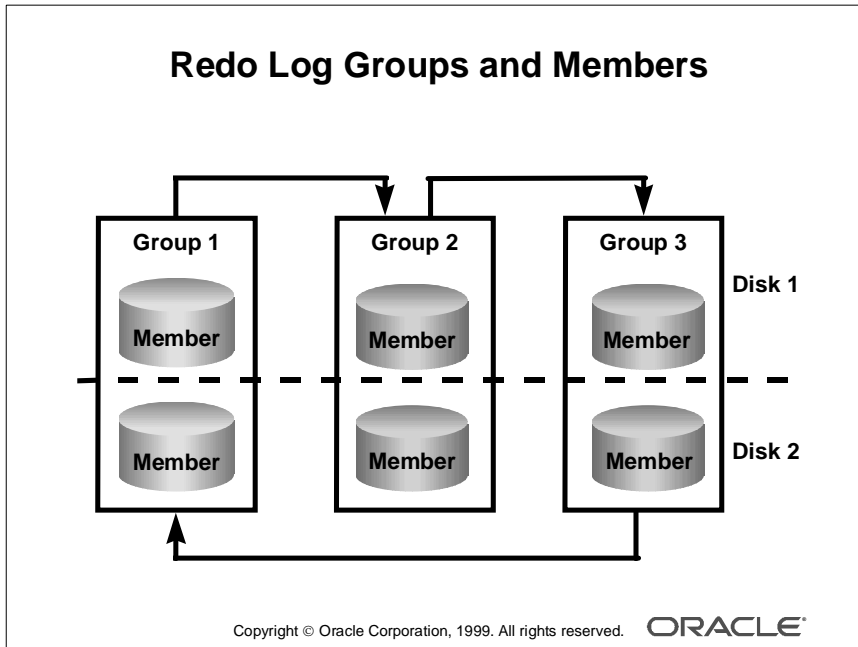


Purposes of the Redo Logs Files

The Oracle server maintains online redo log files to minimize the loss of data in the database. The redo log files record all changes made to data in the database buffer cache with some exceptions; for example, in the case of direct writes.

Redo log files are used in a situation such as an instance failure to recover committed data that has not been written to the data files. The redo log files are used only for recovery.

Using Online Redo Files



Structure of the Redo Log Files

The database administrator can set up the Oracle database to maintain copies of online redo log files to avoid losing database information due to a single point of failure.

Online Redo Log Groups

- A set of identical copies of online redo log files is called an online redo log *group*.
- The LGWR background process concurrently writes the same information to all online redo log files in a group.
- The Oracle server needs a minimum of two online redo log file groups for the normal operation of a database.

Online Redo Log Members

- Each online redo log file in a group is called a *member*.
- Each member in a group has identical log sequence numbers and the same size.
The log sequence number is assigned each time the Oracle server starts writing to a log group to identify each redo log file uniquely. The current log sequence number is stored in the control file and in the header of all data files.

Creating Initial Redo Log Files

The initial set of online redo log groups and members are created during the database creation.

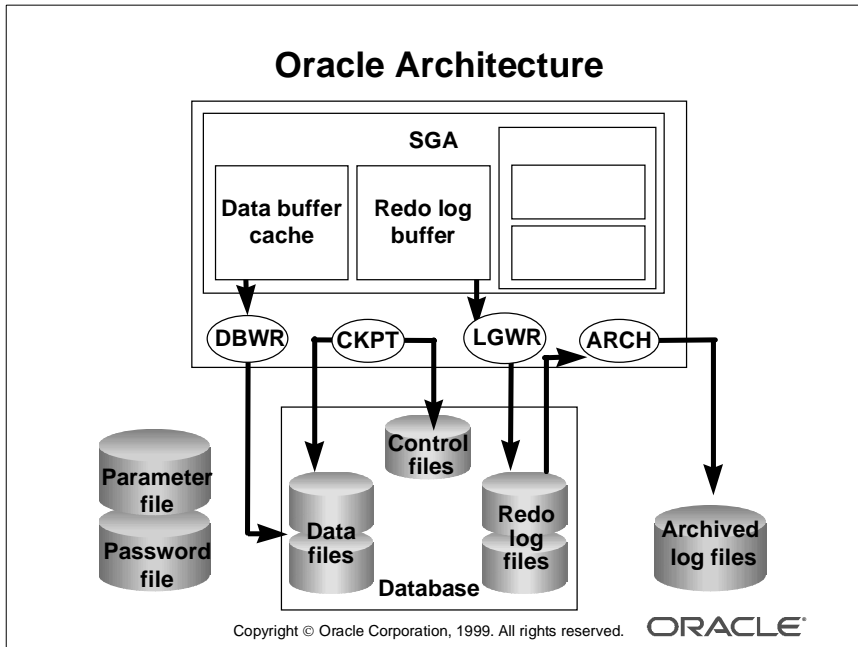
The following parameters limit the number of online redo log files:

- The MAXLOGFILES parameter in the CREATE DATABASE command specifies the absolute maximum of online redo log groups.
The maximum and default value for MAXLOGFILES is dependent on your operating system.
- The MAXLOGMEMBERS parameter used in the CREATE DATABASE command determines the maximum number of members per group. The maximum and default value for MAXLOGMEMBERS is dependent on your operating system.
- The LOG_FILES initialization parameter sets the current maximum number of the log groups that can be opened at run time for the database, and cannot exceed MAXLOGFILES.

Technical Note

The LOG_FILES parameter has been made obsolete in release 8.1 to simplify database administration.

LGWR, Log Switches, and Checkpoints



Redo Log Buffer and the LGWR Background Process

The Oracle server sequentially records all changes made to the database in the redo log buffer. The redo log buffer is used in a circular manner. The redo entries are written to one of the online redo log groups called the *current* online redo log group by the LGWR process under the following situations:

- When a transaction commits
- When the redo log buffer becomes one-third full
- When there is more than a megabyte of changed records in the redo log buffer
- When a timeout occurs (every three seconds)
- Before the DBW_n writes modified blocks in the database buffer cache to the data files

Log Switches

LGWR writes to the online redo log files sequentially—that is, when the current online redo log group is filled, LGWR begins writing to the next group. When the last available online redo log file is filled, LGWR returns to the first online redo log group and starts writing again.

Log Switches (continued)

The database administrator can also force log switches (see subsequent sections). Each time a log switch occurs and LGWR begins writing to a new log group, the Oracle server assigns a number known as the log sequence number to identify the set of redo entries.

When a log switch occurs, an event called a checkpoint is initiated.

A *log switch* is the event during which LGWR stops writing to one online redo log group and starts writing to another.

Checkpoints

During a checkpoint:

- A number of dirty database buffers covered by the log being checkpointed are written to the data files by DBWn. The number of buffers being written by DBWn is determined by the parameter FAST_START_IO_TARGET, if specified. This is covered in more detail in the course *Enterprise DBA Part 1B: Backup and Recovery*.
- The checkpoint background process CKPT updates the headers of all data files and control files to reflect that it has completed successfully.

Checkpoints can occur for *all* data files in the database or for only *specific* data files.

A checkpoint occurs, for example, in the following situations:

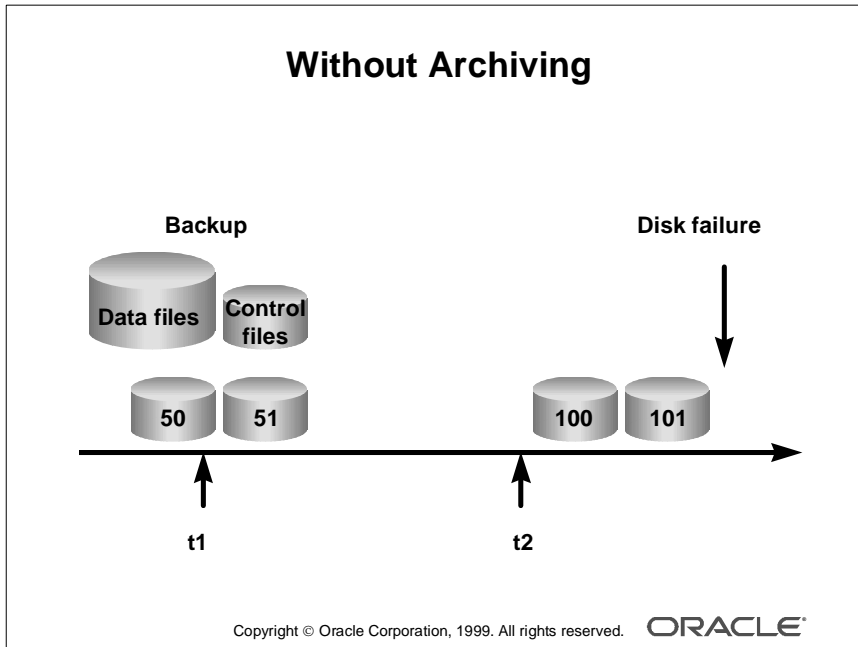
- At every log switch
- When an instance has been shut down with the normal, transactional, or immediate option
- When forced by setting the initialization parameters, LOG_CHECKPOINT_INTERVAL, LOG_CHECKPOINT_TIMEOUT, and FAST_START_IO_TARGET (see subsequent sections)
- When manually requested by the database administrator (see subsequent sections)

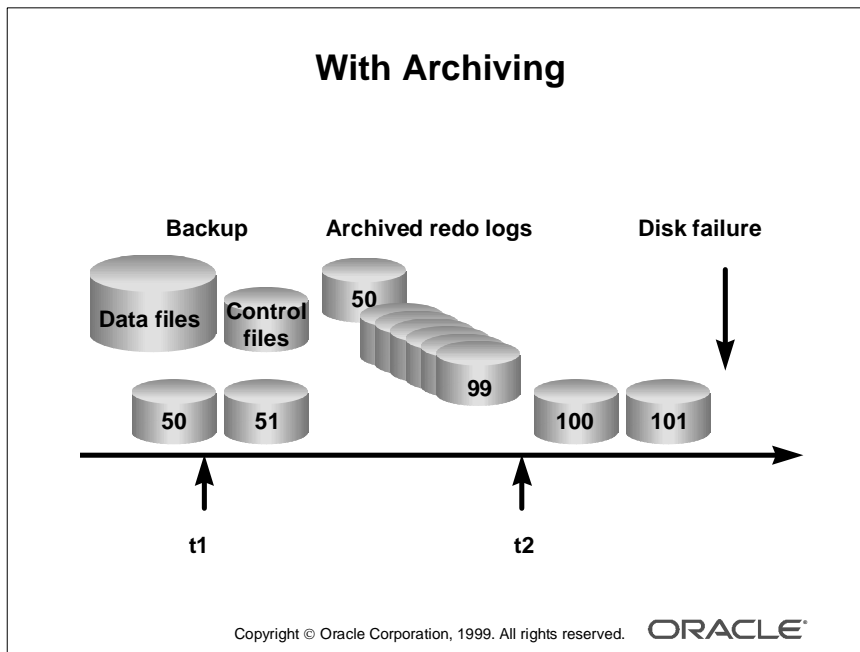
Information about each checkpoint is recorded in the ALERT file if the initialization parameter LOG_CHECKPOINTS_TO_ALERT is set to TRUE. The default value of FALSE for this parameter does not log checkpoints.

Technical Note

The FAST_START_IO_TARGET parameter has been added in release 8.1.

Archiving Redo Log Files





Decide if Archiving of the Redo Log Files Should Be Enabled

One of the important decisions that a database administrator has to make is whether the database is configured to operate in ARCHIVELOG mode or in NOARCHIVELOG mode.

NOARCHIVELOG Mode In NOARCHIVELOG mode, the online redo log files are overwritten each time an online redo log file is filled, and log switches occur. LGWR will not overwrite a redo log group until the checkpoint for that group is completed.

ARCHIVELOG Mode If the database is configured to run in ARCHIVELOG mode, inactive groups of filled online redo log files must be archived. Because all changes made to the database are recorded in the online redo log files, the database administrator can use the physical backup and the archived online redo log files to recover the database without losing any committed data.

There are two ways in which online redo log files can be archived:

- Manually
- Automatically

Decide if Archiving of the Redo Log Files Should Be Enabled (continued)

ARCHIVELOG Mode (continued)

The LOG_ARCHIVE_START initialization parameter indicates whether archiving should be automatic or manual when the instance starts up.

- TRUE indicates that archiving is *automatic*. ARCn will initiate archiving of the filled log group at every log switch.
- FALSE, the default value, indicates that the database administrator will archive filled redo log files *manually*. The database administrator must manually execute a command each time he or she wants to archive an online redo log file. All or specific online redo log files can be archived manually.

Note: Archiving is covered in more detail in the course *Enterprise DBA Part 1B: Backup and Recovery*.

Obtaining Log and Archive Information

Obtaining Information About Archiving

- **SQL command:**

```
ARCHIVE LOG LIST;
```

- **V\$DATABASE:**
 - **NAME**
 - **LOG_MODE**
- **V\$INSTANCE:**
 - **ARCHIVER**

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

Obtaining Archive Information

The following SQL command shows the database log mode and if the automatic archival is enabled:

```
SQL> ARCHIVE LOG LIST
Database log mode                No Archive Mode
Automatic archival               Disabled
Archive destination              ?/dbs/arch
Oldest online log sequence       688
Current log sequence             689
```

Query the dynamic performance views V\$DATABASE and V\$INSTANCE to show the database log mode and the archiving mode.

```
SQL> SELECT name, log_mode
       2 FROM v$database;

NAME          LOG_MODE
-----
U15           NOARCHIVELOG
1 row selected.
```

Obtaining Archive Information (continued)

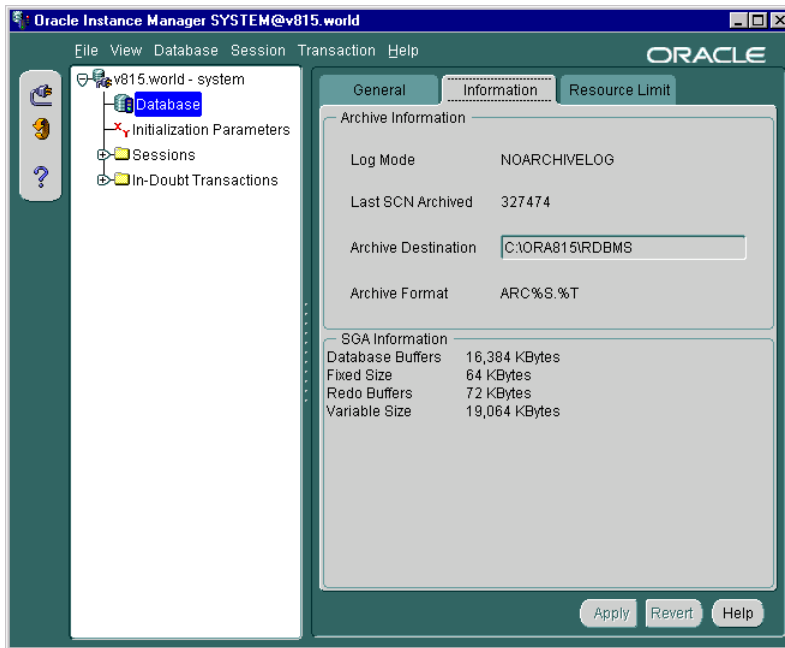
```
SQL>  SELECT archiver  
       2    FROM v$instance;
```

```
ARCHIVE
```

```
-----
```

```
STOPPED
```

```
1 row selected.
```



How to Use Instance Manager to Obtain Archive Information

Launch Instance Manager and obtain information about archiving.

- 1 Launch Instance Manager and connect directly to the database:
Start—>Programs—>Oracle - *EMV2 Home*—>DBA Management Pack
—>Instance Manager
- 2 Enter the login information, and click OK.
- 3 Expand your working database and select Database in the navigator tree.
- 4 Click the Information tab to obtain the archive information.

Obtaining Information About Groups

V\$THREAD:

- GROUPS
- CURRENT_GROUP#
- SEQUENCE#

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Obtaining Log Group Information

To see the number of online redo log groups, the current log group, and the sequence number, query the dynamic performance view V\$THREAD. This is of particular interest to Parallel Server administrators.

```
SQL> SELECT groups, current_group#, sequence#  
2 FROM v$thread;
```

GROUPS	CURRENT_GR	SEQUENCE#
-----	-----	-----
2	1	689

1 row selected.

Obtaining Information About Groups and Members

V\$LOG:

- GROUP#
- MEMBERS
- STATUS
- SEQUENCE#
- BYTES

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE®

Obtaining Log Group Information (continued)

The following query returns information about the online redo log file from the control file:

```
SQL> SELECT group#, sequence#, bytes, members, status
       2 FROM v$log;
```

GROUP#	SEQUENCE#	BYTES	MEMBERS	STATUS
-----	-----	-----	-----	-----
1	688	1048576	1	CURRENT
2	689	1048576	1	INACTIVE

2 rows selected.

The following items are the most common values for the STATUS column:

- **UNUSED** indicates that the online redo log group has never been written to. This is the state of an online redo log file that was just added.
- **CURRENT** indicates the current online redo log group. This implies that the online redo log group is active.
- **ACTIVE** indicates that the online redo log group is active but is not the current online redo log group. It is needed for crash recovery. It may be in use for block recovery. It may or may not be archived.

Obtaining Log Group Information (continued)

- **CLEARING** indicates the log is being re-created as an empty log after an **ALTER DATABASE CLEAR LOGFILE** command. After the log is cleared, the status changes to **UNUSED**.
- **CLEARING_CURRENT** indicates that the current log file is being cleared of a closed thread. The log can stay in this status if there is some failure in the switch, such as an I/O error writing the new log header.
- **INACTIVE** indicates that the online redo log group is no longer needed for instance recovery. It may or may not be archived.

Obtaining Information About Groups and Members

V\$LOGFILE:

- GROUP#
- STATUS
- MEMBER

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Obtaining Log Member Information

To obtain the names of all the members of a group, query the dynamic performance view V\$LOGFILE.

```
SQL> SELECT *
2> FROM v$logfile;
GROUP#STATUSMEMBER
-----
1      /DISK3/log1a.rdo
2      /DISK4/log2a.rdo
```

The value of the STATUS column could be one of the following:

- INVALID indicates that the file is inaccessible.
- STALE indicates that contents of the file are incomplete; for example, adding a log file member.
- DELETED indicates that the file is no longer used.
- Blank indicates that the file is in use.

Controlling Log Switches and Checkpoints

Log Switches and Checkpoints

- Force log switches with the command:

```
ALTER SYSTEM SWITCH LOGFILE;
```

- Control checkpoints with the initialization parameters:
 - LOG_CHECKPOINT_INTERVAL
 - LOG_CHECKPOINT_TIMEOUT
 - FAST_START_IO_TARGET

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE

Forcing Log Switches and Checkpoints

A log switch occurs when the LGWR stops writing to an online redo log group and starts writing to another.

Log switches and checkpoints are events that happen automatically; for example, when the current online log file group is filled. But log switches and checkpoints can be forced.

Forcing Log Switches

You can force a log switch using the following SQL command:

```
SQL> ALTER SYSTEM SWITCH LOGFILE;
```

Forcing Checkpoints

You can manually force a checkpoint by using the following SQL command:

```
SQL> ALTER SYSTEM CHECKPOINT;
```

Setting Database Checkpoint Intervals

When the database uses large online redo log files, you can set additional database checkpoints by setting the initialization parameters:

- `LOG_CHECKPOINT_INTERVAL`
- `LOG_CHECKPOINT_TIMEOUT`
- `FAST_START_IO_TARGET` (in release 8.1, but only for the Enterprise Edition)

LOG_CHECKPOINT_INTERVAL For versions prior to release 8.1, a checkpoint is initiated as soon as the LGWR writes the number of blocks specified by the parameter `LOG_CHECKPOINT_INTERVAL`.

The value of `LOG_CHECKPOINT_INTERVAL` is specified in operating system blocks and not in Oracle database blocks.

Regardless of this value, a checkpoint always occurs when switching from one online redo log file to another.

If the value exceeds the actual online redo log file size, checkpoints occur only when switching logs.

Note that specifying a value of 0 for the interval might cause checkpoints to be initiated very frequently, because a new request will be started even if a single redo log buffer has been written since the last request was initiated.

With release 8.1, when `LOG_CHECKPOINT_INTERVAL` is specified, the target for checkpoint position cannot lag the end of the log more than the number of redo log blocks specified by this parameter. This ensures that no more than a fixed number of redo blocks will need to be read during instance recovery.

LOG_CHECKPOINT_TIMEOUT For versions prior to release 8.1, the value of this initialization parameter specifies the maximum amount of time before another checkpoint occurs. The value is specified in seconds. The time begins at the start of the previous checkpoint, and then a checkpoint occurs after the amount of time specified by this parameter.

Specifying a value of 0 for the timeout disables time-based checkpoints.

With release 8.1, when `LOG_CHECKPOINT_TIMEOUT` is specified, it sets the target for checkpoint position to a location in the log file where the end of the log was this many seconds ago. This ensures that no more than the specified number of seconds' worth of redo blocks needs to be read during recovery.

Setting Database Checkpoint Intervals (continued)

FAST_START_IO_TARGET The parameter FAST_START_IO_TARGET improves the performance of crash and instance recovery. The smaller the value of this parameter, the better the recovery performance, because fewer blocks need to be recovered. When the parameter is set, the DBW n writes dirty buffers out more aggressively. The parameter was introduced in release 8.1.


Technical Note

The fast checkpointing procedure is covered in more details in the course *Enterprise DBA Part 1B: Backup and Recovery*.

Multiplexing and Maintaining Members and Groups

Adding Online Redo Log Groups

```
ALTER DATABASE ADD LOGFILE
('/DISK3/log3a.rdo',
'/DISK4/log3b.rdo') size 1M;
```



Group 1 Group 2 Group 3

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE

Adding Redo Log Groups

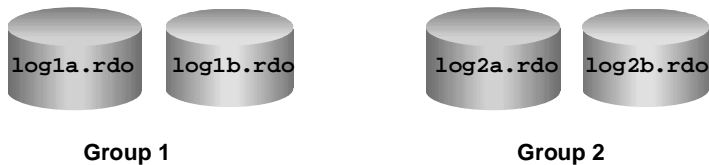
In some cases you might need to create additional log file groups. For example, adding groups can solve availability problems. To create a new group of online redo log files, use the following SQL command:

```
ALTER DATABASE [database]
ADD LOGFILE [GROUP integer] filespec
[, [GROUP integer] filespec]...
```

You specify the name and location of the members with the file specification. The value of the GROUP parameter can be chosen for each redo log file group. If you omit this parameter, the Oracle server generates its value automatically.

Adding Online Redo Log Members

```
ALTER DATABASE ADD LOGFILE MEMBER  
'/DISK4/log1b.rdo' TO GROUP 1,  
'/DISK4/log2b.rdo' TO GROUP 2;
```



Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE®

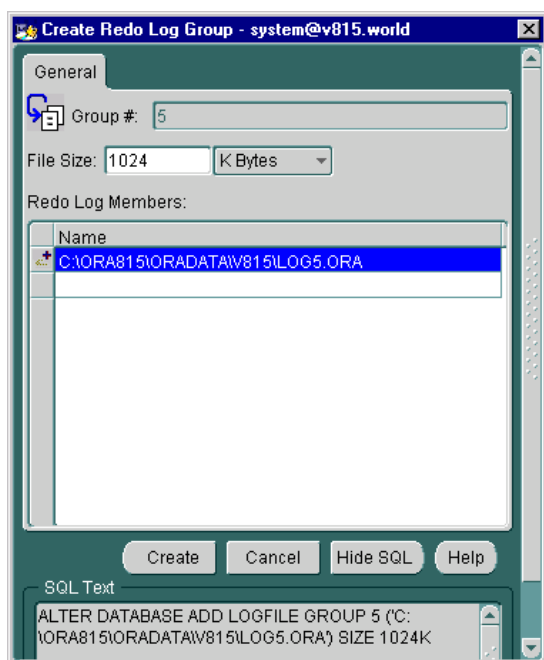
Adding Redo Log Members

You can add new members to existing redo log file groups using the following ALTER DATABASE ADD LOGFILE MEMBER command:

```
ALTER DATABASE [database]  
  ADD LOGFILE MEMBER  
  [  
    'filename' [REUSE]  
    [, 'filename' [REUSE]]...  
  TO {GROUP integer  
      | ('filename' [, 'filename']...)  
      }  
  ]...
```

Use the fully specified name of the log file members; otherwise, the files will be created in a default directory of the database server.

If the file already exists, it must have the same size and you must specify the REUSE option. You can identify the target group either by specifying one or more members of the group or by specifying the group number.



How to Use Storage Manager to Maintain Groups and Members

Launch Storage Manager to manage redo log groups and members.

- 1 Launch Storage Manager and connect directly to the database:
Start—>Programs—>Oracle - *EMV2 Home*—>DBA Management Pack
—>Storage Manager
- 2 Enter the login information, and click OK.
- 3 Expand your working database and select the Redo Log Groups folder in the navigator tree. Click the right mouse button and select Create.
- 4 Enter your redo log group information and specify the members. Click Create.

Relocating or Renaming Online Redo Log Files

How to Relocate or Rename Online Redo Log Files

1. Copy the online redo log files to the new location.
2. Execute the **ALTER DATABASE RENAME FILE** command.

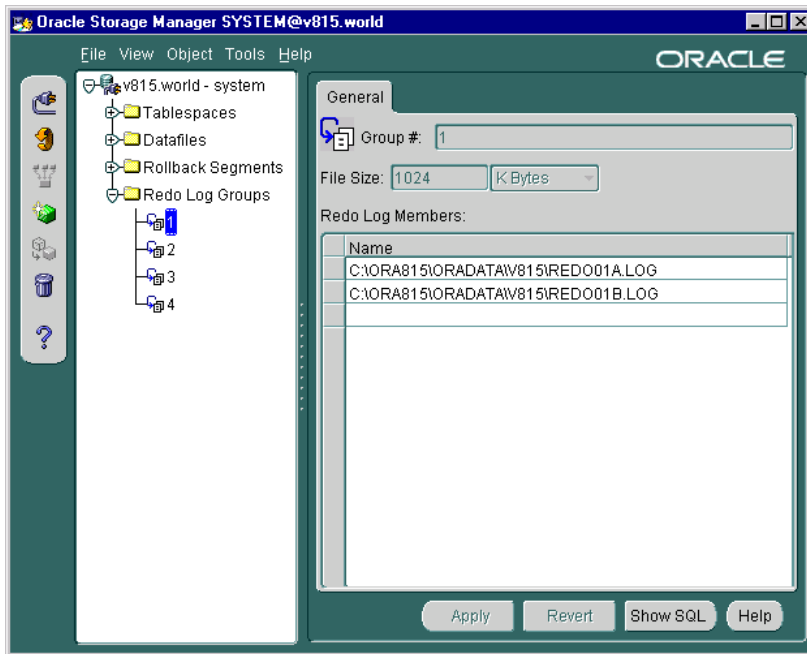
Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

Renaming Redo Log Files

The locations of the online redo log files can be changed by renaming the online redo log files. Before renaming the online redo log files, ensure that the new online redo log file exists. The Oracle server changes only the pointers in the control files, but does not physically rename or create any operating system files.

The following **ALTER DATABASE RENAME FILE** command changes the name of the online redo log file:

```
ALTER DATABASE [database]
  RENAME FILE 'filename'[, 'filename']...
  TO 'filename'[, 'filename']...
```

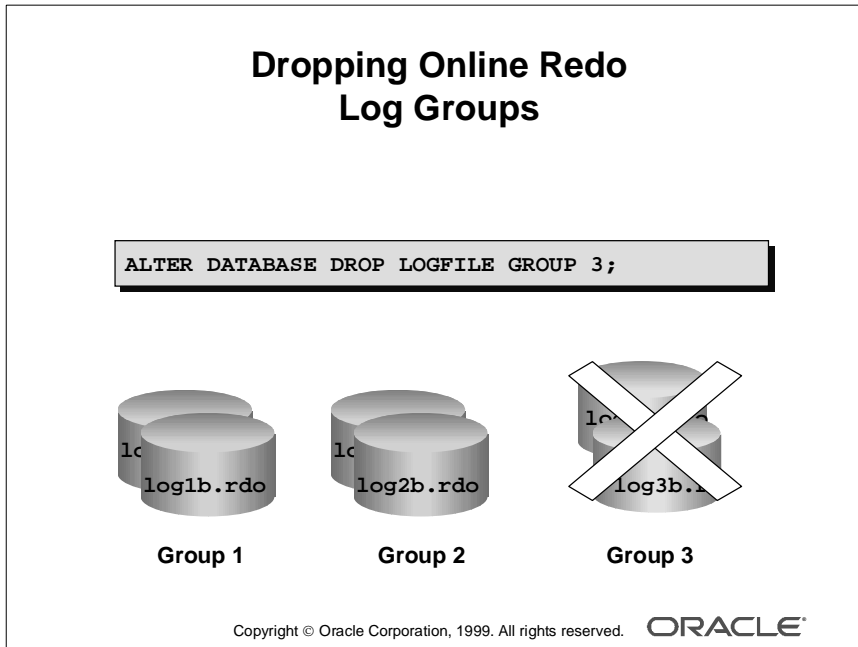



How to Use Storage Manager to Relocate or Rename Redo Log Members

Launch Storage Manager to rename or relocate redo log members.

- 1 Launch Storage Manager and connect directly to the database:
Start—>Programs—>Oracle - *EMV2 Home*—>DBA Management Pack
—>Storage Manager
- 2 Enter the login information, and click OK.
- 3 Expand your working database and select a redo log group.
- 4 Modify your redo log member information to rename or relocate members. Click Apply.

Dropping Online Redo Log Groups and Members



Dropping a Redo Log Group

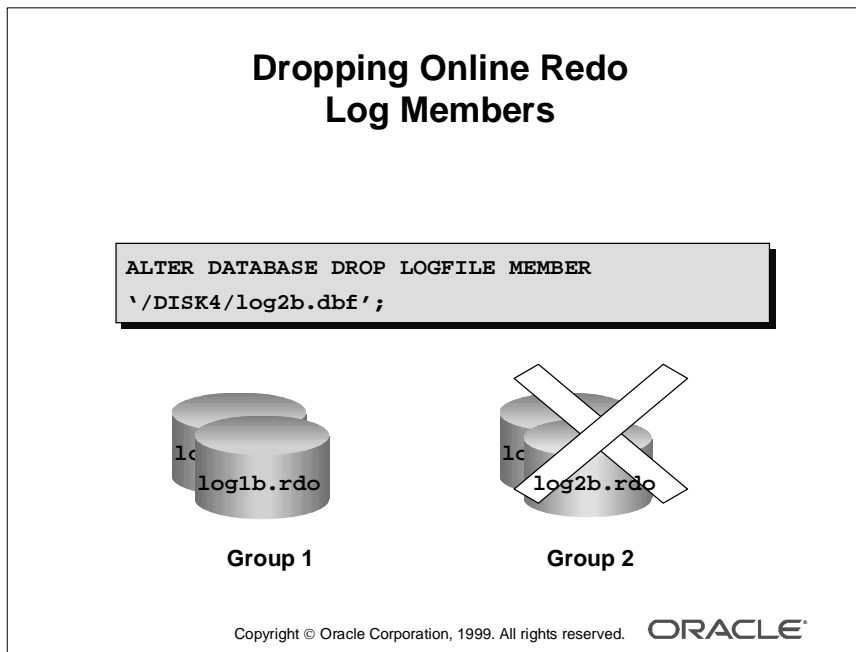
To increase or decrease the size of online redo log groups, add new online redo log groups (with the new size) and then drop the old ones.

An entire online redo log group can be dropped with the following ALTER DATABASE DROP LOGFILE command:

```
ALTER DATABASE [database]
  DROP LOGFILE
    {GROUP integer|('filename'[, 'filename']...)}
    [, {GROUP integer|('filename'[, 'filename']...)}]...
```

Restrictions

- An instance requires at least two groups of online redo log files.
- An active or current group cannot be dropped.
- If the database is running in ARCHIVELOG mode and the log file group is not archived, then the group cannot be dropped.
- When an online redo log group is dropped, the operating system files are not deleted.



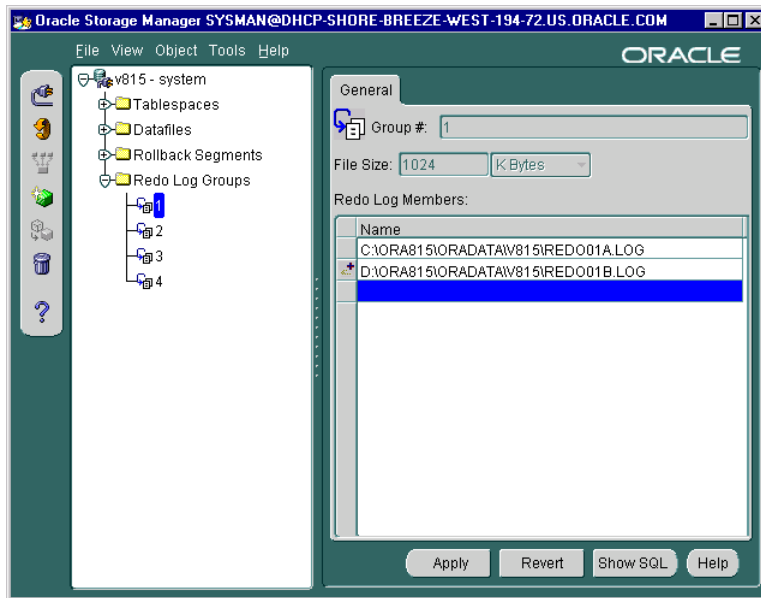
Dropping a Redo Log Member

You may want to drop an online redo log member because it has a status of **INVALID**. Use the following **ALTER DATABASE DROP LOGFILE MEMBER** command if you want to drop one or more specific online redo log members:

```
ALTER DATABASE [database]
DROP LOGFILE MEMBER 'filename'[, 'filename']...
```

Restrictions

- If the member you want to drop is the last valid member of the group, you cannot drop that member.
- If the group is current, you must force a log file switch before you can drop the member.
- If the database is running in **ARCHIVELOG** mode and the log file group to which the member belongs is not archived, then the member cannot be dropped.
- When an online redo log member is dropped, the operating system file is not deleted.



How to Use Storage Manager to Drop Redo Log Groups and Members

Launch Storage Manager to drop redo log groups and members.

- 1 Launch Storage Manager and connect directly to the database:
Start—>Programs—>Oracle - *EMV2 Home*—>DBA Management Pack
—>Storage Manager
- 2 Enter the login information, and click OK.
- 3 Expand your working database and select a redo log group or a redo log member, depending on which object you want to remove.
- 4 Select Object—>Remove from the menu bar to remove a redo log group or a member.

Clearing Online Redo Log Files

Example:

```
ALTER DATABASE CLEAR LOGFILE  
  '/DISK3/log2a.rdo';
```

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Clearing Online Redo Log Files

If a redo log file is corrupted in all members, the database administrator can solve this problem by reinitializing these log files.

The SQL command `ALTER DATABASE CLEAR LOGFILE` reinitializes online redo log files:

```
ALTER DATABASE [database]  
  CLEAR [UNARCHIVED] LOGFILE  
    {GROUP integer|('filename'[, 'filename']...)}  
    [, {GROUP integer|('filename'[, 'filename']...)}]...
```

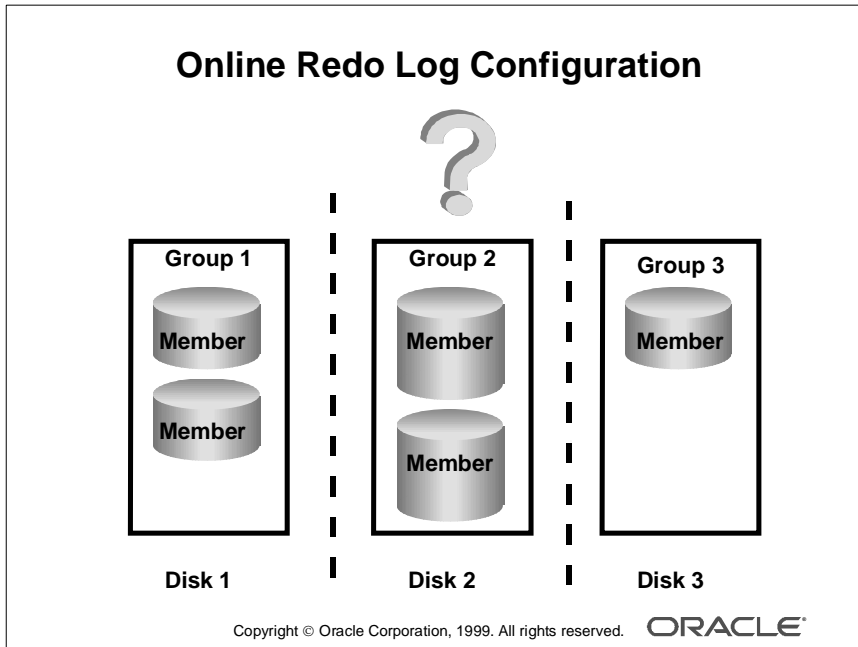
Using this command is equivalent to adding and dropping an online redo log file. But you can issue this command even if there are only two log groups with one file each, and even if the cleared group is available but not archived.

Restrictions

You can clear an online redo log file whether it is archived or not. However, when it is not archived, you must include the keyword `UNARCHIVED`. This will make backups unusable if the online redo log file is needed for recovery.

Note: Clearing online redo log files is covered in more detail in the course *Enterprise DBA Part 1B: Backup and Recovery*.

Planning Online Redo Logs



Number of Online Redo Log Files

To determine the appropriate number of online redo log files for a database instance, you have to test different configurations.

In some cases, a database instance may require only two groups. In other situations, a database instance may require additional groups to guarantee that the groups are always available to LGWR. For example, if messages in the LGWR trace file or in the ALERT file indicate that LGWR frequently has to wait for a group because a checkpoint has not completed or a group has not been archived, you need to add groups.

Although the Oracle server allows multiplexed groups to contain different numbers of members, try to build up a symmetric configuration. An asymmetric configuration should only be the temporary result of an unusual situation such as a disk failure.

Location of Online Redo Log Files

When you multiplex the online redo log files, place members of a group on different disks. By doing this, even if one member is not available but other members are available, the instance does not shut down.

Separate archive log files and online redo log files on different disks to reduce contention between the ARC*n* and LGWR background processes.

Location of Online Redo Log Files (continued)

Data files and online redo log files should be placed on different disks to reduce LGWR and DBW n contention and reduce the risk of losing both data files and online redo log files in the event of media failure.

Sizing Online Redo Log Files

The minimum size of an online redo log file is 50 KB and the maximum size is specific to the operating system. Members of different groups can have different sizes; however, there is no benefit to having different-sized groups.

Different-sized groups should only be required as a temporary result if you want to change the size of the members of the online redo log groups. In this case, you have to create new online redo log groups with different sizes, and then remove the old groups.

The following situations might influence the configuration of the online redo log files:

- Number of log switches and checkpoints
- Number and amount of redo entries
- Amount of space on the storage medium; for example, on a tape if archiving is enabled

Troubleshooting

Possible LGWR Errors

- One member of a group of two or more is not available.
- All members of the next group are not available.
- All members of the current group are not available.

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE

Unavailable Redo Log Members

The LGWR reacts differently when certain online redo log members are unavailable.

- If LGWR can access at least one member in a group, the writing to the accessible members of the group proceeds as usual; LGWR ignores the unavailable members of the group. If the group was not active—that is, the checkpoint was completed—then dropping and adding a new redo log member solves the problem. Otherwise, you have to first force a log switch.
- If all members of the next group are inaccessible to LGWR at a log switch, the instance shuts down. If the group was not active, then dropping and adding a new redo log group solves the problem. If not, the database may need media recovery from the loss of an online redo log file.
- If all members of the current group suddenly become inaccessible to LGWR as they are being written, the database instance shuts down. In this case, the database may need media recovery from the loss of an online redo log file.

Using LogMiner

Analyzing the Redo Log Files

- **Track changes:**
 - To the database
 - To a specific table
 - To a specific user
- **Map data access patterns**
- **Undo changes to the database**
- **Use archived data to perform tuning and capacity planning**

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

What Can LogMiner Be Used For?

LogMiner provides a procedure to process the redo log files and translate their contents into SQL statements that represent the logical operations performed to the database.

Technical Note

LogMiner runs in Oracle release 8.1 or later.

Redo log files from any release 8.0 or later of the database can be analyzed.

How to Use LogMiner

- Specify UTL_FILE_DIR
- Create a dictionary file

```
EXECUTE DBMS_LOGMNR_D.BUILD('v815dict.ora',  
'C:\ora815\admin\v815\log');
```

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

What You Should Do Before Using LogMiner

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

Without a dictionary file, the equivalent SQL statements will use an Oracle internal object ID for the object name and present column values as hex data.

Creating a Dictionary File

- Specify the initialization parameter UTL_FILE_DIR to specify a directory that is permitted for PL/SQL file I/O.
- Execute the DBMS_LOGMNR_D.BUILD procedure to create the dictionary file.

Specifying Log Files to Be Analyzed

Set up the V\$LOGMNR_CONTENTS view:

- Initialize a new list and specify the first log file.

```
EXECUTE DBMS_LOGMNR.ADD_LOGFILE(  
'c:\ora815\oradata\v815\redo01a.log',  
DBMS_LOGMNR.NEW);
```

- Add additional log files to the list.

```
EXECUTE DBMS_LOGMNR.ADD_LOGFILE(  
'c:\ora815\oradata\v815\redo02a.log',  
DBMS_LOGMNR.ADDFILE);
```

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Setting Up the LogMiner Session

Once you have created a dictionary file, you can begin analyzing redo logs. The first step is to specify the log files that you want to analyze, using the DBMS_LOGMNR.ADD_LOGFILE procedure.

Use the following constants:

- DBMS_LOGMNR.NEW creates a new list and specifies the first log file.
- DBMS_LOGMNR.ADDFILE adds additional log files to the list.
- DBMS_LOGMNR.REMOVEFILE removes redo logs from the list.

LogMiner can analyze both online and archived log files.

Starting to Analyze the Redo Log Files

Initialize a LogMiner session:

```
EXECUTE DBMS_LOGMNR.START_LOGMNR(  
  DICTFILENAME=>  
    'c:\ora815\oradata\v815\log\v815dict.ora');
```

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Starting a LogMiner Session

Once a dictionary file has been created and a list of redo logs has been specified, you can start LogMiner and begin your analysis. Use the following options to narrow down the search at start time:

Option	Description
StartScn	The beginning of a system change number (SCN) range
EndScn	The termination of an SCN range
StartTime	The beginning of a time interval
EndTime	The end of a time interval
DictFileName	The name of the dictionary file
Options	Use the column map specified in <code>logmnr.opt</code> file; the value is <code>USE_COLMAP</code>

Tracking Changes to a Table

View V\$LOGMNR_CONTENTS to track changes for the EMP table:

```
SELECT timestamp, username, sql_redo
FROM v$logmnr_contents
WHERE seg_name = 'EMP';
```

TIMESTAMP	USER	SQL_REDO
14-APR-99	SYS	update SCOTT.EMP set sal =..
14-APR-99	SYS	update SCOTT.EMP set sal =..
14-APR-99	SYS	insert into SCOTT.EMP(...)...

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Tracking Changes to a Table

View the output by way of the V\$LOGMNR_CONTENTS view. Only the session that performed the analysis can view the log information. Other sessions cannot see the information. If the results are to be viewed by other sessions, it is useful to store the information in another table.

Finishing Analysis of the Redo Log Files

Finish the LogMiner session:

```
EXECUTE DBMS_LOGMNR.END_LOGMNR;
```

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

Stopping the LogMiner Session

Execute the DBMS_LOGMNR.END_LOGMNR procedure to finish the session analyzing the redo logs.

Obtaining Information About Logs Being Analyzed

- V\$LOGMNR_DICTIONARY
- V\$LOGMNR_PARAMETERS
- V\$LOGMNR_CONTENTS

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE®

Viewing the Data Dictionary

Once LogMiner has been started, the following data dictionary views can be used:

View	Description
V\$LOGMNR_DICTIONARY	The dictionary file in use
V\$LOGMNR_PARAMETERS	Current parameter settings for the LogMiner
V\$LOGMNR_CONTENTS	The contents of the redo log files being analyzed

Summary

Summary

In this lesson, you should have learned how to:

- **Explain the use of online redo log files**
- **Obtain log and archive information**
- **Control log switches and checkpoints**
- **Multiplex and maintain online redo log files**
- **Plan online redo log files**
- **Troubleshoot common redo log file problems**
- **Analyze online and archived redo logs**

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

Quick Reference

Context	Reference
Initialization parameters	LOG_FILES (Obsolete in release 8.1) LOG_CHECKPOINTS_TO_ALERT UTL_FILE_DIR
Dynamic initialization parameters	LOG_CHECKPOINT_TIMEOUT LOG_CHECKPOINT_INTERVAL FAST_START_IO_TARGET
Dynamic performance views	V\$THREAD V\$LOG V\$LOGFILE V\$DATABASE V\$LOGMNR_CONTENTS V\$LOGMNR_DICTIONARY V\$LOGMNR_LOGS V\$LOGMNR_PARAMETERS
Data dictionary views	None
Commands	ALTER SYSTEM SWITCH LOGFILE ALTER SYSTEM CHECKPOINT ARCHIVE LOG LIST ALTER DATABASE ADD LOGFILE ALTER DATABASE ADD LOGFILE MEMBER ALTER DATABASE RENAME FILE ALTER DATABASE DROP LOGFILE ALTER DATABASE DROP LOGFILE MEMBER ALTER DATABASE CLEAR LOGFILE
Packaged procedures and functions	DBMS_LOGMNR.D.BUILD DBMS_LOGMNR.ADD_LOGFILE DBMS_LOGMNR.START_LOGMNR DBMS_LOGMNR.END_LOGMNR

8

Managing Tablespaces and Data Files

Objectives

Objectives

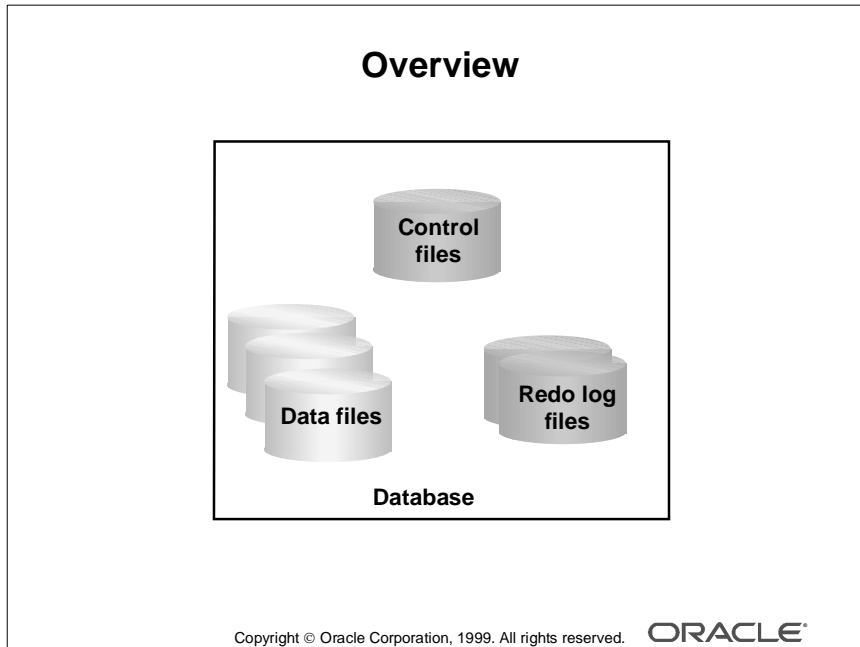
After completing this lesson, you should be able to do the following:

- Describe the logical structure of the database
- Distinguish the different types of temporary segments
- Create tablespaces
- Change the size of tablespaces
- Allocate space for temporary segments
- Change the status of tablespaces
- Change the storage settings of tablespaces
- Relocate tablespaces

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Overview

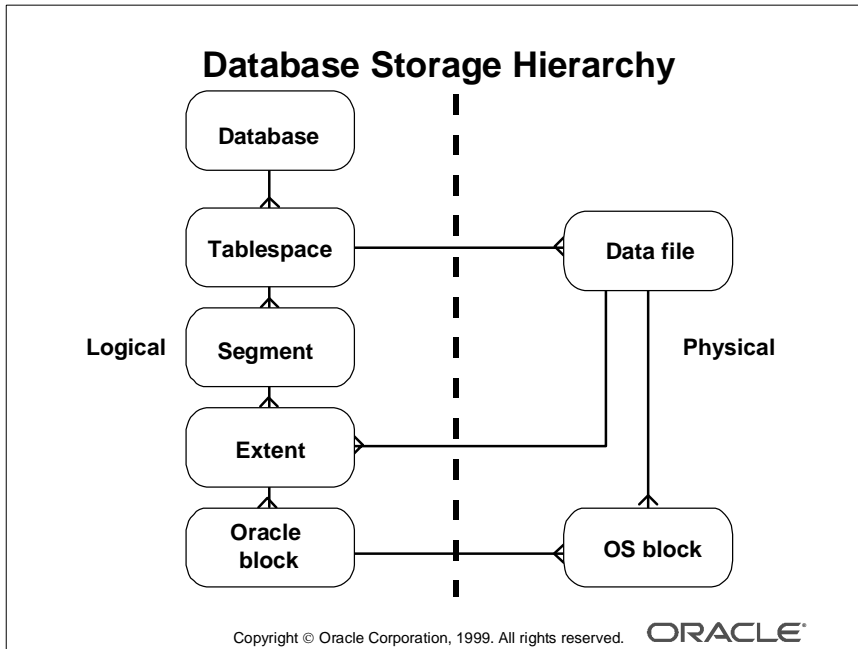


Overview

A small database might need only the SYSTEM tablespace; however, Oracle recommends that you create additional tablespaces to store user data, user indexes, rollback segments, and temporary segments separate from data dictionary. This gives you more flexibility in various database administration operations and reduces contention among dictionary objects and schema objects for the same data files.

The DBA can create new tablespaces, resize data files, add data files to tablespaces, set and alter default segment storage settings for segments created in a tablespace, make a tablespace read-only or read-write, make a tablespace temporary or permanent, and drop tablespaces.

Database Storage Hierarchy



Database Architecture

The Oracle database architecture includes logical and physical structures that make up the database.

- The physical structure includes the control files, online redo log files, and data files that make up the database.
- The logical structure includes tablespaces, segments, extents, and data blocks.

The Oracle server enables fine-grained control of disk space use through tablespace and logical storage structures, including segments, extents, and data blocks.

Tablespaces

The data in an Oracle database are stored in tablespaces.

- An Oracle database can be logically grouped into smaller logical areas of space known as tablespaces.
- A tablespace can belong to only one database at a time.
- Each tablespace consists of one or more operating system files, which are called data files.

Tablespaces (continued)

- A tablespace may consist of one or more segments.
- Tablespaces can be brought online while the database is running.
- Except for the SYSTEM tablespace or a tablespace with an active rollback segment, tablespaces can be taken offline, leaving the database running.
- Tablespaces can be switched between read-write and read-only status.

Data Files

Each tablespace in an Oracle database consists of one or more files called data files. These are physical structures that conform with the operating system on which the Oracle server is running.

- A data file can belong to only one tablespace.
- An Oracle server creates a data file for a tablespace by allocating the specified amount of disk space plus a small amount of overhead.
- The database administrator can change the size of a data file after its creation or can specify that a data file should dynamically grow as objects in the tablespace grow.

Segments

A segment is the space allocated for a specific logical storage structure within a tablespace. For example, all of the storage allocated to a table is a segment.

- A tablespace may consist of one or more segments.
- A segment cannot span tablespaces; however, a segment can span multiple data files that belong to the same tablespace.
- Each segment is made up of one or more extents.

Extents

Space is allocated to a segment by extents.

- One or more extents make up a segment.
 - When a segment is created, it consists of at least one extent.
 - As the segment grows, extents are added to the segment.
 - The DBA can manually add extents to a segment.
- An extent is a set of contiguous Oracle blocks.
- An extent may not span a data file, but must exist in one data file.

Data Blocks

The Oracle server manages the storage space in the data files in units called Oracle blocks or data blocks.

- At the finest level of granularity, the data in an Oracle database is stored in data blocks.
- Oracle data blocks are the smallest units of storage that the Oracle server can allocate, read, or write.
- One data block corresponds to one or more operating system blocks allocated from an existing data file.
- Data block size is specified for each Oracle database by the initialization parameter `DB_BLOCK_SIZE` when the database is created.
- The data block size should be a multiple of the operating system block size to avoid unnecessary I/O.
- The maximum data block size is dependent on the operating system.

SYSTEM and Non-SYSTEM Tablespaces

SYSTEM and Non-SYSTEM Tablespaces

- **SYSTEM tablespace:**
 - Created with the database
 - Contains the data dictionary
 - Contains the SYSTEM rollback segment
- **Non-SYSTEM tablespaces:**
 - Separate segments
 - Ease space administration
 - Control amount of space allocated to a user

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Types of Tablespaces

The DBA creates tablespaces for increased control and ease of maintenance. The Oracle server perceives two types of tablespaces: SYSTEM and all others.

SYSTEM Tablespace

- Created with the database
- Required in all databases
- Contains the data dictionary, including stored program units
- Contains the SYSTEM rollback segment
- Should not contain user data, although it is allowed

Non-SYSTEM Tablespaces

- Enable more flexibility in database administration
- Separate rollback, temporary, application data, and application index segments
- Separate data by backup requirements
- Separate dynamic and static data
- Control the amount of space allocated to user's objects

Creating Tablespaces

Creating Tablespaces

```
CREATE TABLESPACE app_data
  DATAFILE '/DISK4/app_data_01.dbf'
           SIZE 100M,
           '/DISK5/app_data_02.dbf'
           SIZE 100M
  MINIMUM EXTENT 500K
  DEFAULT STORAGE ( INITIAL      500K
                   NEXT         500K
                   MAXEXTENTS   500
                   PCTINCREASE   0 );
```

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

CREATE TABLESPACE Command

You create a tablespace with the CREATE TABLESPACE command:

```
CREATE TABLESPACE tablespace
  DATAFILE datafile_clause]
  [, datafile_clause]...
  [MINIMUM EXTENT integer[K|M]]
  [LOGGING|NOLOGGING]
  [DEFAULT storage_clause ]
  [ONLINE|OFFLINE]
  [PERMANENT|TEMPORARY]
  [extent_management_clause]
```

where:	tablespace	is the name of the tablespace to be created
	DATAFILE	specifies the data file or data files that make up the tablespace
	MINIMUM EXTENT	ensures that every used extent size in the tablespace is a multiple of the <i>integer</i> . (Use K or M to specify this size in kilobytes or megabytes.)

CREATE TABLESPACE Command (continued)

LOGGING	specifies that, by default, all tables, indexes, and partitions within the tablespace will have all changes written to redo (LOGGING is the default.)
NOLOGGING	specifies that, by default, all tables, indexes, and partitions within the tablespace will not have all changes written to redo (NOLOGGING only affects some DML and DDL commands; for example, direct loads.)
DEFAULT	specifies the default storage parameters for all objects created in the tablespace
ONLINE	makes the tablespace available for use immediately upon creation
OFFLINE	makes the tablespace unavailable immediately after creation
PERMANENT	specifies that the tablespace can be used to hold permanent objects
TEMPORARY	specifies that the tablespace will only be used to hold temporary objects; for example, segments used by implicit sorts caused by an ORDER BY clause
extent_management_clause	specifies how the extents of the tablespace will be managed (This clause is discussed in a subsequent section of this lesson.)

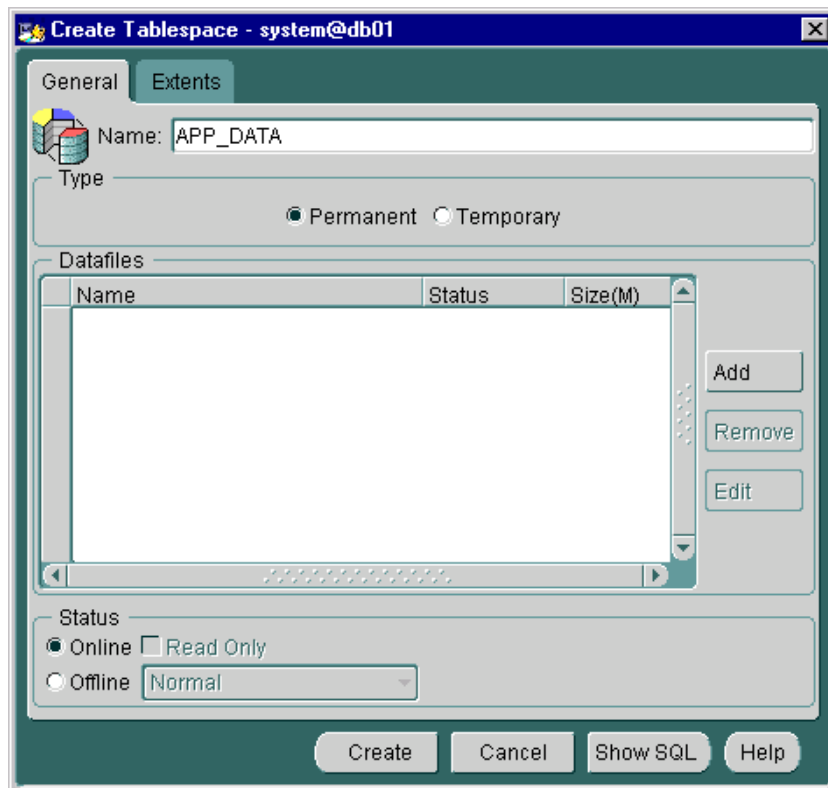
CREATE TABLESPACE Command (continued)

```
datafile_clause ::= filename  
                  {SIZE integer[K|M] [REUSE] | REUSE }  
                  [ autoextend_clause ]
```

where:	filename	is the name of a data file in the tablespace
	SIZE	specifies the size of the file (Use K or M to specify the size in kilobytes or megabytes.)
	REUSE	allows the Oracle server to reuse an existing file
	autoextend_clause	enables or disables the automatic extension of the data file (This clause is discussed in a subsequent section of this lesson.)

How to Use Oracle Enterprise Manager to Create a New Tablespace

- 1 Launch Storage Manager and connect directly to the database:
Start—>Programs—>Oracle - *EMV2 Home*—>DBA Management Pack
—>Storage Manager
- 2 Enter the login information, and click OK.
- 3 Select the Tablespaces folder, and choose Create from the right mouse menu.
- 4 In the General page of the property sheet, enter the name and click ADD to display the Create Data file property sheet.



- 5 In the Create Data file property sheet, specify each data file.
- 6 In the Extents page of the property sheet, enter storage information.
- 7 Click Create.

Space Management in Tablespaces

Space Management in Tablespaces

- **Dictionary-managed tablespaces:**
 - **Default technique**
 - **Free extents recorded in data dictionary tables**
- **Locally managed tablespaces:**
 - **Free extents recorded in bitmap**
 - **Each bit corresponds to a block or group of blocks**
 - **Bit value indicates free or used**

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

Choosing a Space Management Method

Tablespace extents can be managed with data dictionary tables or bitmaps. When you create a tablespace, you choose one of these methods of space management. You cannot alter the method at a later time.

Dictionary-Managed Tablespaces

For a tablespace that uses the data dictionary to manage its extents, the Oracle server updates the appropriate tables in the data dictionary whenever an extent is allocated or deallocated.

This is the default method of space management in a tablespace. It is the only method available in Oracle release 8.0 and earlier.

Locally Managed Tablespaces

A tablespace that manages its own extents maintains a bitmap in each data file to keep track of the free or used status of blocks in that data file. Each bit in the bitmap corresponds to a block or a group of blocks. When an extent is allocated or freed for reuse, the Oracle server changes the bitmap values to show the new status of the blocks.

Locally Managed Tablespaces

Locally Managed Tablespaces

```
CREATE TABLESPACE user_data
  DATAFILE '/DISK2/user_data_01.dbf'
  SIZE 500M
  EXTENT MANAGEMENT LOCAL
  UNIFORM SIZE 10M;
```

- Reduced recursive space management
- Reduced contention on data dictionary tables
- No rollback generated
- No coalescing required

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Syntax

The LOCAL option of the EXTENT MANAGEMENT clause specifies that a tablespace is to be locally managed.

extent_management_clause ::=

```
[ EXTENT MANAGEMENT
  { DICTIONARY | LOCAL
    { AUTOALLOCATE | UNIFORM [SIZE integer[K|M]] } } ]
```

where:	DICTIONARY	specifies that the tablespace is managed using dictionary tables (This is the default.)
	LOCAL	specifies that tablespace is locally managed with a bitmap
	AUTOALLOCATE	specifies that the tablespace is system managed (Users cannot specify extent size.)

Syntax (continued)

UNIFORM	specifies that the tablespace is managed with uniform extents of SIZE bytes (Use K or M to specify the extent size in kilobytes or megabytes. The default SIZE is 1 megabyte. If you specify LOCAL, you cannot specify DEFAULT storage_clause, MINIMUM EXTENT, or TEMPORARY)
---------	--

The EXTENT MANAGEMENT clause can be used in various CREATE commands:

- For a permanent tablespace other than SYSTEM, you can specify EXTENT MANAGEMENT LOCAL in the CREATE TABLESPACE command.
- For a temporary tablespace, you can specify EXTENT MANAGEMENT LOCAL in the CREATE TEMPORARY TABLESPACE command.

Advantages of Locally Managed Tablespaces

Locally managed tablespaces have the following advantages over dictionary-managed tablespaces:

- Local management avoids recursive space management operations, which can occur in dictionary-managed tablespaces if consuming or releasing space in an extent results in another operation that consumes or releases space in a rollback segment or data dictionary table.
- Because locally managed tablespaces do not record free space in data dictionary tables, it reduces contention on these tables.
- Local management of extents automatically tracks adjacent free space, eliminating the need to coalesce free extents.
- The sizes of extents that are managed locally can be determined automatically by the system. Alternatively, all extents can have the same size in a locally managed tablespace.
- Changes to the extent bitmaps do not generate rollback information because they do not update tables in the data dictionary (except for special cases such as tablespace quota information).

Temporary Tablespace

Temporary Tablespace

- Used for sort operations
- Cannot contain any permanent objects
- Locally managed extents recommended
- **UNIFORM SIZE = SORT_AREA_SIZE * n**

```
CREATE TEMPORARY TABLESPACE temp
  TEMPFILE '/DISK2/temp_01.dbf'
  SIZE 500M
  EXTENT MANAGEMENT LOCAL
  UNIFORM SIZE 10M;
```

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Temporary Segments

You can manage space for sort operations more efficiently by designating temporary tablespaces exclusively for sort segments. No permanent schema objects can reside in a temporary tablespace.

Sort, or temporary, segments are used when a segment is shared by multiple sort operations. Temporary tablespaces provide performance improvements when you have multiple sorts that are too large to fit into memory. The sort segment of a given temporary tablespace is created at the time of the first sort operation of the instance. The sort segment expands by allocating extents until the segment size is equal to or greater than the total storage demands of all of the active sorts running on that instance.

CREATE TEMPORARY TABLESPACE Command

Although the ALTER/CREATE TABLESPACE...TEMPORARY command can be used to create a temporary tablespace, it is recommended that you use the CREATE TEMPORARY TABLESPACE command.

CREATE TEMPORARY TABLESPACE Command (continued)

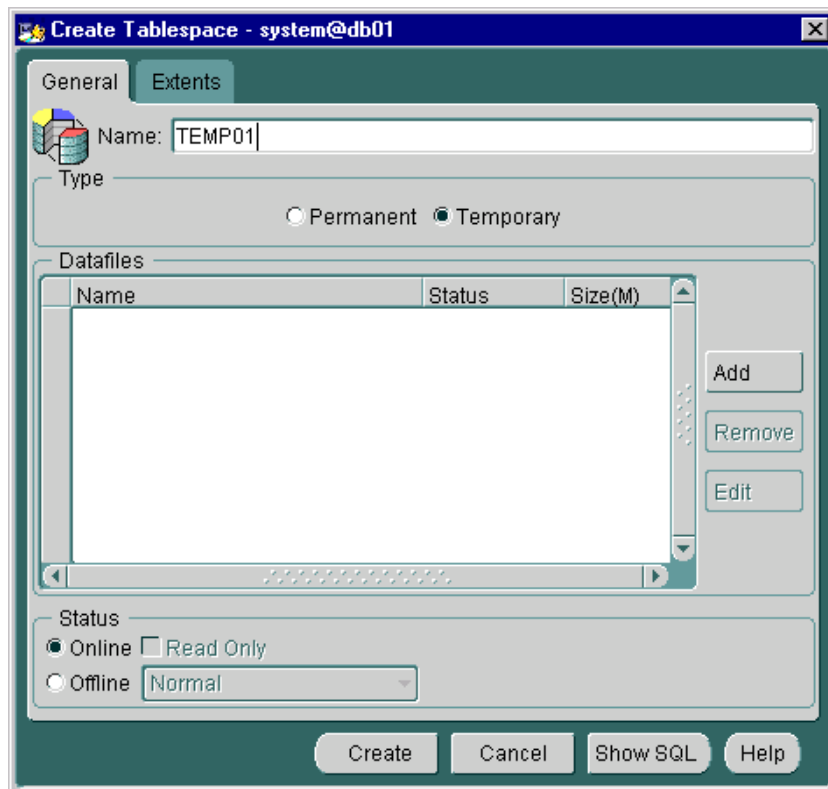
Locally managed temporary tablespaces have temporary data files (tempfiles), which are similar to ordinary data files except that:

- Tempfiles are always set to NOLOGGING mode.
- You cannot make a tempfile read-only.
- You cannot rename a tempfile.
- You cannot create a tempfile with the ALTER DATABASE command.
- Media recovery does not recover tempfiles.
- BACKUP CONTROLFILE does not generate any information for tempfiles.
- CREATE CONTROLFILE cannot specify any information about tempfiles.

To optimize the performance of a sort in a temporary tablespace, set the UNIFORM SIZE to be a multiple of the parameter SORT_AREA_SIZE.

How to Use Oracle Enterprise Manager to Create a Temporary Tablespace

- 1 Launch Storage Manager and connect directly to the database:
Start—>Programs—>Oracle - *EMV2 Home*—>DBA Management Pack
—>Storage Manager
- 2 Enter the login information, and click OK.
- 3 Select the Tablespaces folder, and choose Create from the right mouse menu.
- 4 In the General page of the property sheet, enter the name and select the TEMPORARY option button.



- 5 Click ADD to display the Create Data file property sheet.
- 6 In the Create Data file property sheet, specify each data file.
- 7 Click Create.

Changing the Storage Settings

Changing the Storage Settings

```
ALTER TABLESPACE app_data  
  MINIMUM EXTENT 2M;
```

```
ALTER TABLESPACE app_data  
  DEFAULT STORAGE (  
    INITIAL      2M  
    NEXT         2M  
    MAXEXTENTS 999 );
```

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

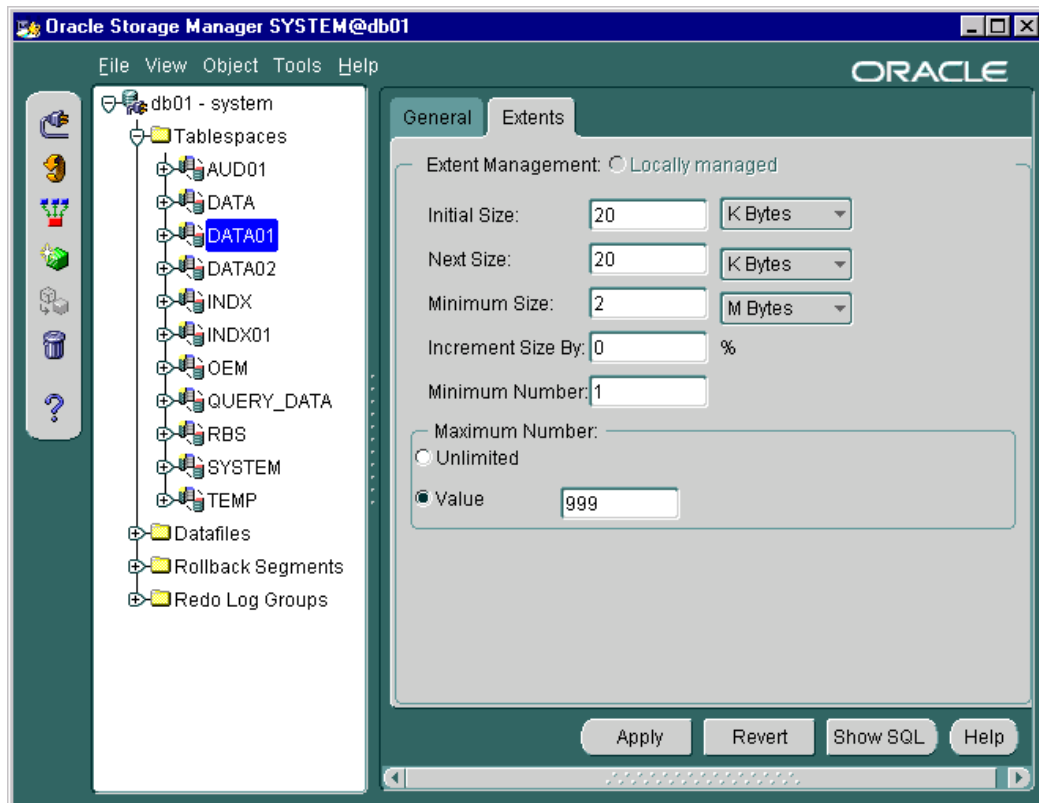
Changing Default Storage Settings

Use the ALTER TABLESPACE command to alter the default storage definition of a tablespace.

```
ALTER TABLESPACE tablespace  
  { MINIMUM EXTENT integer[K|M]  
    | DEFAULT storage_clause }
```

How to Use Oracle Enterprise Manager to Change the Storage Settings

- 1 Launch Storage Manager and connect directly to the database:
Start—>Programs—>Oracle - *EMV2 Home*—>DBA Management Pack
—>Storage Manager
- 2 Enter the login information, and click OK.
- 3 Expand the Tablespaces folder.
- 4 Select the tablespace.
- 5 In the Extents page of the property sheet, enter storage information.



- 6 Click Apply.

Taking Tablespaces Offline or Online

Offline Status

- Offline tablespace not available for data access
- Some tablespaces must be online:
 - SYSTEM
 - Tablespaces with active rollback segments
- To take a tablespace offline:

```
ALTER TABLESPACE app_data OFFLINE;
```

- To bring a tablespace online:

```
ALTER TABLESPACE app_data ONLINE;
```

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Taking a Tablespace Offline

A tablespace is typically online so that the data contained within it is available to database users. However, the database administrator might take a tablespace offline to:

- Make a portion of the database unavailable, while allowing normal access to the remainder of the database
- Perform an offline tablespace backup (although a tablespace can be backed up while online and in use)
- Recover a tablespace or data file while the database is open
- Move a data file while the database is open

The Offline Status of a Tablespace

When a tablespace goes offline, the Oracle server does not permit any subsequent SQL statements to reference objects contained in that tablespace. Users trying to access objects in a tablespace that is offline receive an error.

When a tablespace goes offline or comes back online, the event is recorded in the data dictionary and in the control file. If a tablespace is offline when you shut down a database, the tablespace remains offline and will not be checked when the database is subsequently mounted and reopened.

The Offline Status of a Tablespace (continued)

The Oracle instance automatically switches a tablespace from online to offline when certain errors are encountered (for example, when the Database Writer process, DBW0, fails in several attempts to write to a data file of the tablespace). The different error situations are covered in more detail in the course *Enterprise DBA Part 1B: Backup and Recovery*.

Taking Tablespaces Offline

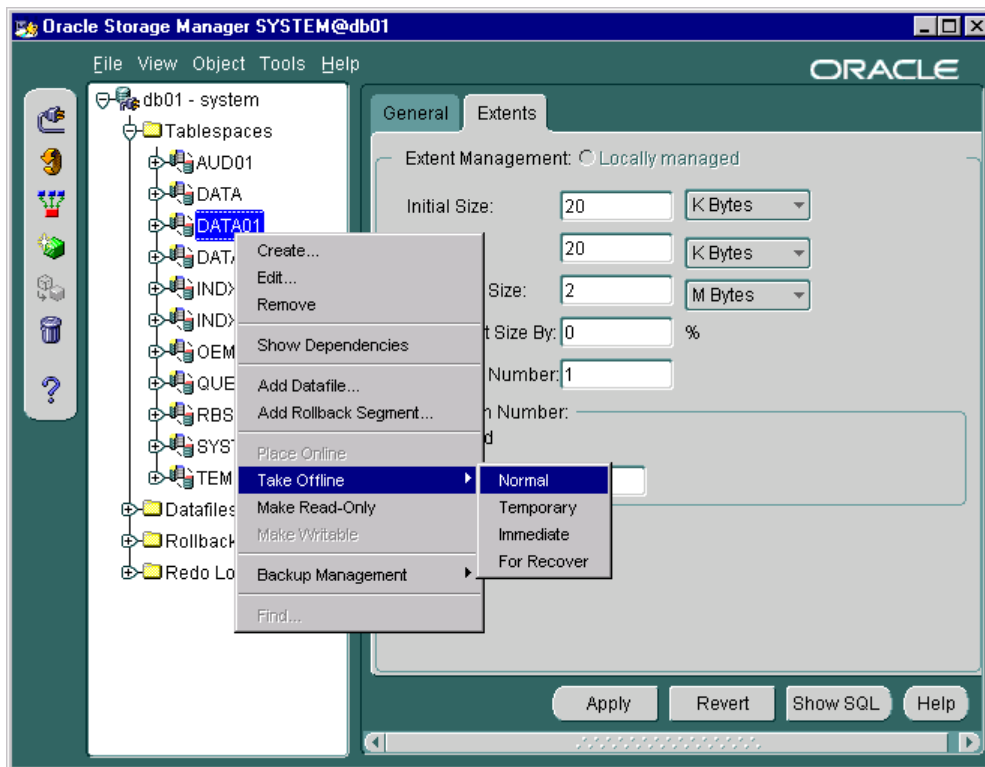
Whenever the database is open, a database administrator can take any tablespace offline, except the SYSTEM tablespace or any tablespace with active rollback segments or temporary segments. When a tablespace is taken offline, the Oracle server takes all the associated data files offline.

```
ALTER TABLESPACE tablespace
    { ONLINE
    | OFFLINE [NORMAL | TEMPORARY | IMMEDIATE | FOR RECOVER] }
```

where:	NORMAL	flushes all blocks in all data files in the tablespace out of the SGA (This is the default. You need not perform media recovery on this tablespace before bringing it back online. Use the NORMAL clause whenever possible.)
	TEMPORARY	performs a checkpoint for all online data files in the tablespace only (Any offline files may require media recovery.)
	IMMEDIATE	does not ensure that tablespace files are available and does not perform a checkpoint (You must perform media recovery on the tablespace before bringing it back online.)
	FOR RECOVER	takes tablespaces offline for tablespace point-in-time recovery

How to Use Oracle Enterprise Manager to Take a Tablespace Offline

- 1 Launch Storage Manager and connect directly to the database:
Start—>Programs—>Oracle - *EMV2 Home*—>DBA Management Pack
—>Storage Manager
- 2 Enter the login information, and click OK.
- 3 Expand the Tablespaces folder.
- 4 Select the tablespace.
- 5 Select Take Offline—>*Mode*.



- 6 Click Yes in the dialog box.

Read-Only Tablespaces

Read-Only Tablespaces

```
ALTER TABLESPACE app_data READ ONLY;
```

- Tablespace only available for read operations
- Objects can be dropped from tablespace
- To create a read-only tablespace on a WORM drive:
 - ALTER TABLESPACE...READ ONLY;
 - Move the data file to the WORM drive
 - ALTER TABLESPACE...RENAME DATAFILE...;

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

The ALTER TABLESPACE...READ ONLY Command

Making tablespaces read-only prevents further write operations on the data files in the tablespace. Therefore, the data files can reside on read-only media, such as CD-ROMs or write-once (WORM) drives. Read-only tablespaces eliminate the need to perform backups of large, static portions of a database. Use the ALTER TABLESPACE SQL command to change a tablespace to read-only or read-write.

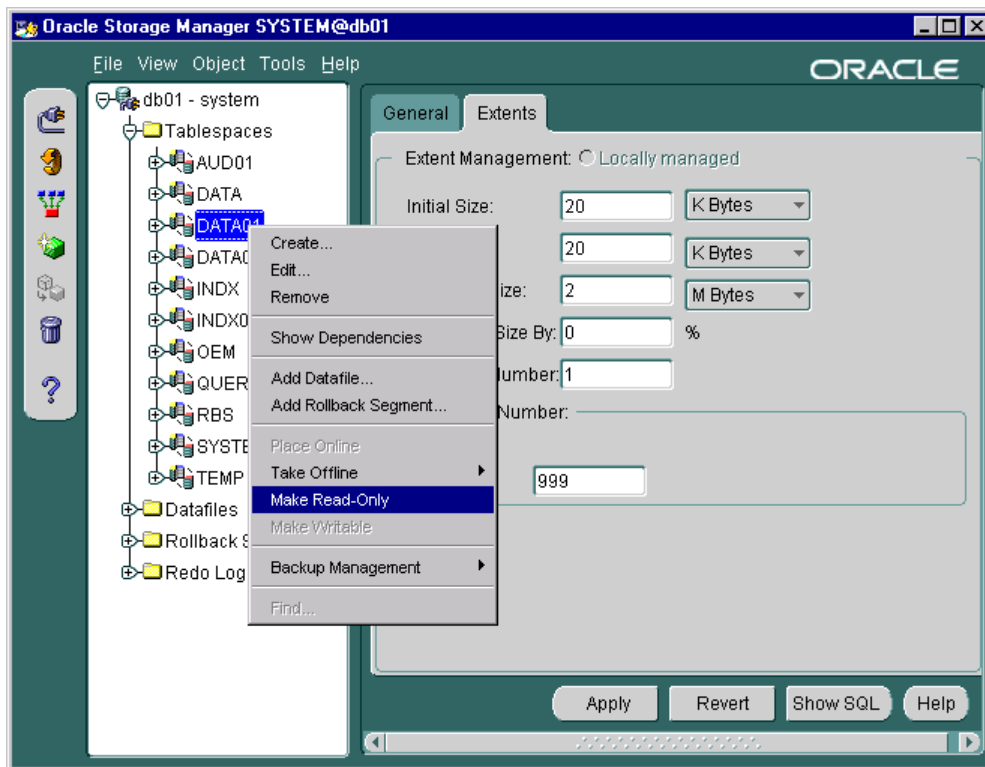
```
ALTER TABLESPACE tablespace READ [ONLY | WRITE]
```

To create a read-only tablespace on a write-once device:

- 1 Issue the command ALTER TABLESPACE...READ ONLY.
- 2 Use an operating system command to move the data files of the tablespace to the read-only device.
- 3 Issue the command ALTER TABLESPACE...RENAME DATAFILE.

How to Use Oracle Enterprise Manager to Make a Tablespace Read-Only

- 1 Launch Storage Manager and connect directly to the database:
Start—>Programs—>Oracle - *EMV2 Home*—>DBA Management Pack
—>Storage Manager
- 2 Enter the login information, and click OK.
- 3 Expand the Tablespaces folder.
- 4 Select the tablespace.
- 5 Select Make Read-Only from the right mouse menu.



- 6 Click OK.

Making a Tablespace Read-Only

- **The tablespace must be online.**
- **In releases prior to Oracle8i, no active transactions are allowed.**
- **Oracle8i allows current transactions to complete.**
- **The tablespace must not contain active rollback segments.**
- **The tablespace must not currently be involved in an online backup.**

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

Making Tablespaces Read-Only

In Oracle8i, the ALTER TABLESPACE ... READ ONLY command places the tablespace in a transitional read-only mode. This transitional state does not allow any further write operations to the tablespace except for the rollback of existing transactions that previously modified blocks in the tablespace. After all of the existing transactions have either committed or rolled back, the ALTER TABLESPACE ... READ ONLY command completes and the tablespace is placed in read-only mode.

For releases prior to Oracle8i, it is recommended that the DBA start the instance in restricted mode when making a tablespace read-only, because a tablespace cannot be made read-only when there are active transactions.

You can drop items, such as tables and indexes, from a read-only tablespace, because these commands only affect the data dictionary. This is possible because the DROP command only updates the data dictionary, not the physical files that make up the tablespace. For locally managed tablespaces, the dropped segment is changed to a temporary segment, to prevent the bitmap from being updated.

To make a read-only tablespace writable, all of the data files in the tablespace must be online.

Making tablespaces read-only will cause a checkpoint on the data files of the tablespace.

Dropping Tablespaces

Dropping Tablespaces

- **Tablespace removed from data dictionary**
- **Optionally, contents removed from data dictionary**
- **OS files not deleted**

```
DROP TABLESPACE app_data INCLUDING CONTENTS;
```

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

DROP TABLESPACE Command

You can remove a tablespace from the database when the tablespace and its contents are no longer required with the following DROP TABLESPACE SQL command:

```
DROP TABLESPACE tablespace  
[INCLUDING CONTENTS [CASCADE CONSTRAINTS]]
```

where: tablespace specifies the name of the tablespace to be dropped

 INCLUDING CONTENTS drops all the segments in the tablespace

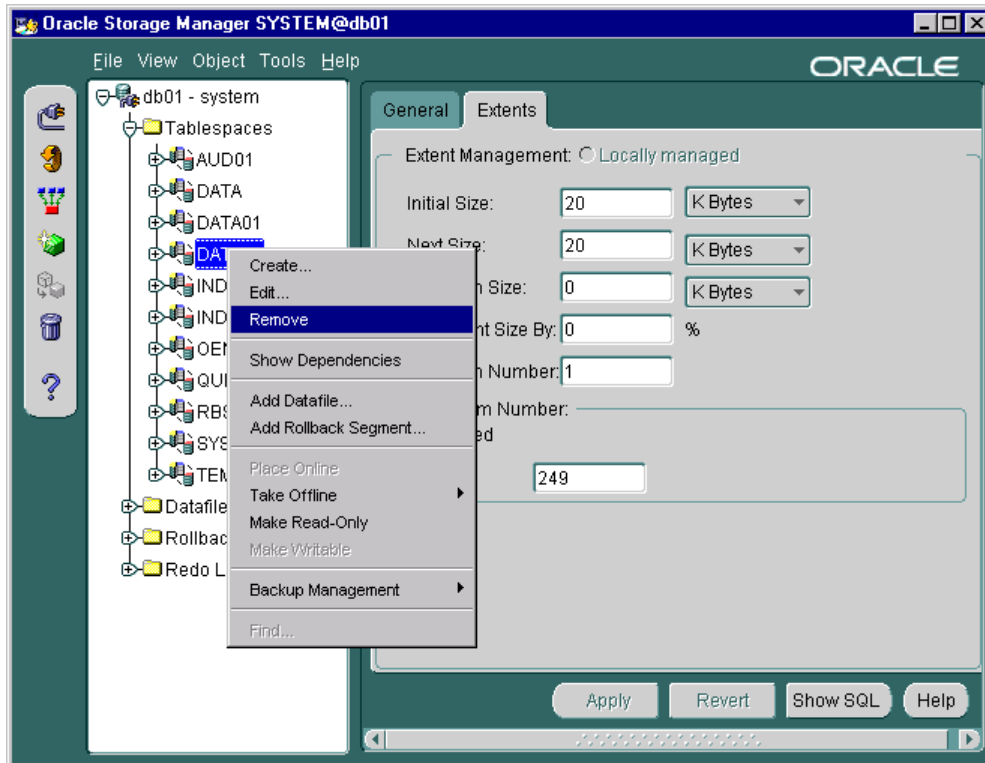
 CASCADE CONSTRAINTS drops referential integrity constraints from
 tables outside the tablespace that refer to
 primary and unique keys in the tables in the
 dropped tablespace

Guidelines

- A tablespace that still contains data cannot be dropped without the INCLUDING CONTENTS option. This option may generate a lot of rollback when the tablespace contains many objects.
- Once a tablespace has been dropped, its data is no longer in the database.
- When a tablespace is dropped, only the file pointers in the control file of the associated database are dropped. The operating system files still exist and must be deleted explicitly using the appropriate operating system command.
- Even if a tablespace is switched to read-only it can still be dropped, along with segments within it.
- It is recommended that you take the tablespace offline before dropping it to ensure that no transactions access any of the segments in the tablespace.

How to Use Oracle Enterprise Manager to Drop a Tablespace

- 1 Launch Storage Manager and connect directly to the database:
Start—>Programs—>Oracle - *EMV2 Home*—>DBA Management Pack
—>Storage Manager
- 2 Enter the login information, and click OK.
- 3 Expand the Tablespaces folder and select the tablespace.
- 4 Select Remove from the right mouse menu.

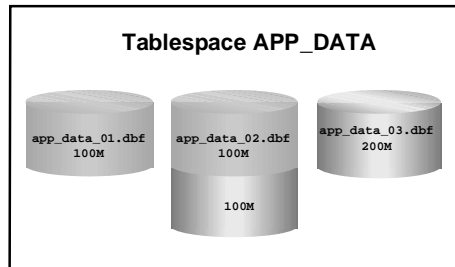


- 5 Click Yes in the dialog box to confirm.

Resizing a Tablespace

Resizing a Tablespace

- Change the size of a data file:
 - Automatically
 - Manually
- Add a data file



Copyright © Oracle Corporation, 1999. All rights reserved.

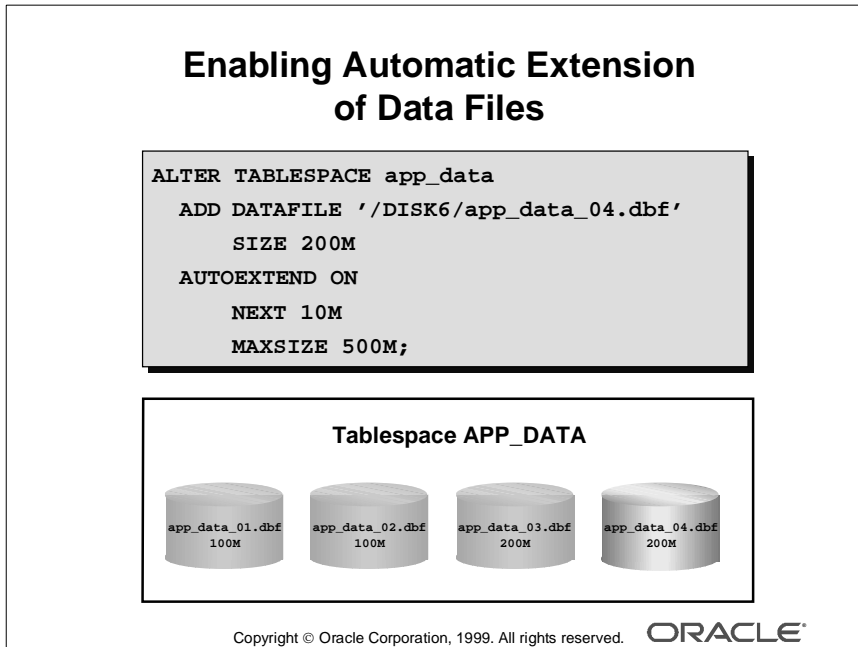
ORACLE®

Increasing the Tablespace Size

You can enlarge a tablespace in two ways:

- Change the size of a data file, either automatically or manually
- Add a data file to a tablespace

Enabling Automatic Resizing of Data Files



Specifying AUTOEXTEND for a New Data File

The AUTOEXTEND clause enables or disables the automatic extension of data files. When a data file is created, the following SQL commands can be used to enable automatic extension of the data file:

- CREATE DATABASE
- CREATE TABLESPACE ... DATAFILE
- ALTER TABLESPACE ... ADD DATAFILE

Use the ALTER TABLESPACE command to add a data file with automatic extension enabled.

```
ALTER TABLESPACE tablespace
  ADD DATAFILE filespec [autoextend_clause]
  [, filespec [autoextend_clause]]...
```


Specifying AUTOEXTEND for a New Data File (continued)

```
autoextend_clause ::= [ AUTOEXTEND { OFF|ON[NEXT integer[K|M]]  
[MAXSIZE UNLIMITED | integer[K|M]] } ]
```

where:	AUTOEXTEND OFF	disables the automatic extension of the data file
	AUTOEXTEND ON	enables the automatic extension of the data file
	NEXT	specifies the disk space to allocate to the data file when more extents are required
	MAXSIZE	specifies the maximum disk space allowed for allocation to the data file
	UNLIMITED	sets no limit on allocating disk space to the data file

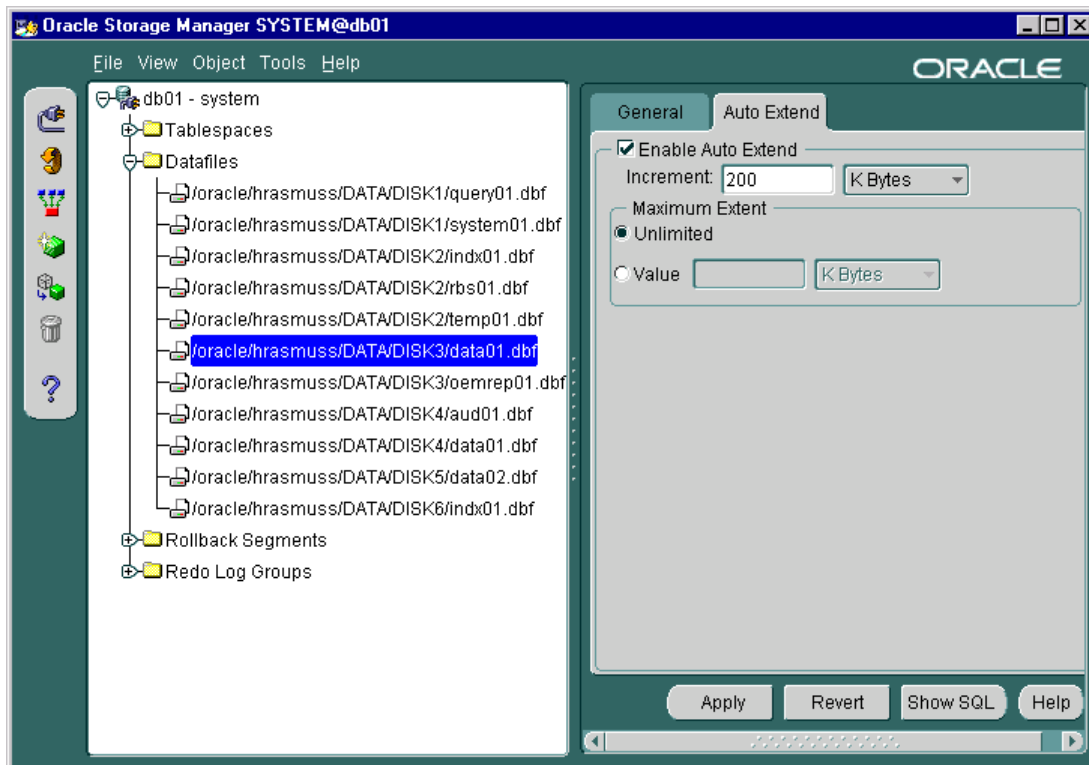
Specifying AUTOEXTEND for an Existing Data File

Use the SQL command ALTER DATABASE to enable or disable automatic file extension for existing data files.

```
ALTER DATABASE [database]  
DATAFILE 'filename'[, 'filename']...autoextend_clause
```

How to Use Oracle Enterprise Manager to Enable Automatic Resizing

- 1 Launch Storage Manager and connect directly to the database:
Start—>Programs—>Oracle - *EMV2 Home*—>DBA Management Pack
—>Storage Manager
- 2 Enter the login information, and click OK.
- 3 Expand the Data files folder.
- 4 Select the data file.
- 5 In the Auto Extend tab of the property sheet, select the Enable Auto Extend check box.

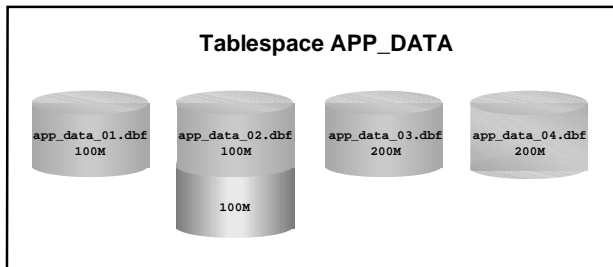


- 6 Click Apply.

Manually Resizing Data Files

Changing the Size of Data Files Manually

```
ALTER DATABASE  
  DATAFILE '/DISK5/app_data_02.dbf'  
  RESIZE 200M;
```



Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

The ALTER DATABASE DATAFILE RESIZE Command

Instead of adding space to the database by adding data files, the DBA can change the size of a data file. Use the ALTER DATABASE command to manually increase or decrease the size of a data file.

```
ALTER DATABASE [database]  
  DATAFILE 'filename'[, 'filename']...  
  RESIZE integer[K|M]
```

where: integer is the absolute size, in bytes, of the resulting data file


If there are database objects stored above the specified size, then the data file size is decreased only to the last block of the last objects in the data file.


Adding Data Files to a Tablespace


Adding Data Files to a Tablespace

```
ALTER TABLESPACE app_data
  ADD DATAFILE '/DISK5/app_data_03.dbf'
  SIZE 200M;
```

Tablespace APP_DATA


app_data_01.dbf
100M


app_data_02.dbf
100M


app_data_03.dbf
200M

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

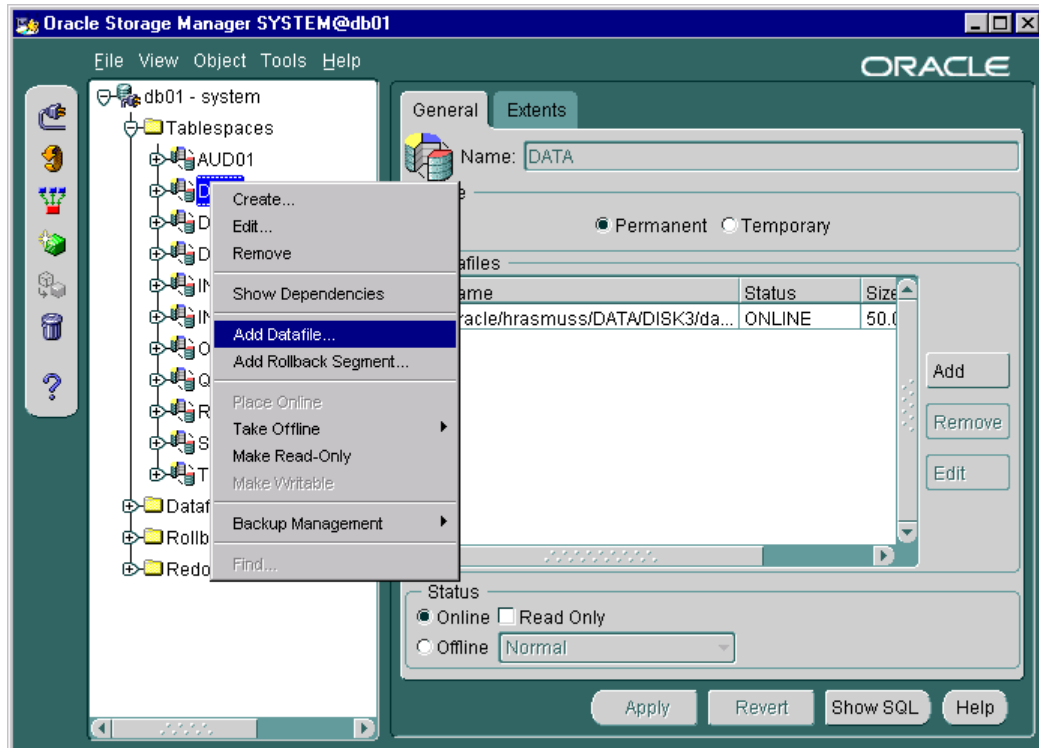
Using the ALTER TABLESPACE ADD DATAFILE Command

You can add data files to a tablespace to increase the total amount of disk space allocated for the tablespace with the ALTER TABLESPACE ADD DATAFILE command.

```
ALTER TABLESPACE tablespace
  ADD DATAFILE    filespec [autoextend_clause]
  [,              filespec [autoextend_clause]]...
```

How to Use Oracle Enterprise Manager to Add a Data File

- 1 Launch Storage Manager and connect directly to the database:
Start—>Programs—>Oracle - *EMV2 Home*—>DBA Management Pack
—>Storage Manager
- 2 Enter the login information, and click OK.
- 3 Expand the Tablespaces folder.
- 4 Select Tablespace—>Add Datafile.



- 5 In the General page of the property sheet, enter the file information.
- 6 Click Create.

Moving Data Files

Moving Data Files: ALTER TABLESPACE

- The tablespace must be offline.
- The target data files must exist.

```
ALTER TABLESPACE app_data
  RENAME
    DATAFILE '/DISK4/app_data_01.dbf'
  TO        '/DISK5/app_data_01.dbf';
```

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Methods for Moving Data Files

Depending on the type of tablespace, the database administrator can move data files using one of two methods: the ALTER TABLESPACE command or the ALTER DATABASE command.

Using the ALTER TABLESPACE Command

The following ALTER TABLESPACE command is applied only to data files in a non-SYSTEM tablespace that does not contain active rollback or temporary segments:

```
ALTER TABLESPACE tablespace
  RENAME DATAFILE 'filename'[, 'filename']...
  TO 'filename'[, 'filename']...
```

Use the following process to rename a data file:

- 1 Take the tablespace offline.
- 2 Use an operating system command to move or copy the files.
- 3 Execute the ALTER TABLESPACE RENAME DATAFILE command.
- 4 Bring the tablespace online.
- 5 Use an operating system command to delete the file if necessary.

The source filenames must match the names stored in the control file.

Moving Data Files: ALTER DATABASE

- The database must be mounted.
- The target data file must exist.

```
ALTER DATABASE  
  RENAME FILE '/DISK1/system_01.dbf'  
  TO '/DISK2/system_01.dbf';
```

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

Using the ALTER DATABASE Command

The ALTER DATABASE command (see the lesson “Maintaining Redo Log Files”) can be used to move any type of data file.

```
ALTER DATABASE [database]  
  RENAME FILE 'filename'[, 'filename']...  
  TO 'filename'[, 'filename']...
```

Because the SYSTEM tablespace cannot be taken offline, you must use this method to move data files in the SYSTEM tablespace.

Use the following process to rename files in tablespaces that cannot be taken offline:

- 1 Shut down the database.
- 2 Use an operating system command to move the files.
- 3 Mount the database.
- 4 Execute the ALTER DATABASE RENAME FILE command.
- 5 Open the database.

How to Use Oracle Enterprise Manager to Add a Data File

- 1** Launch Storage Manager and connect directly to the database:
Start—>Programs—>Oracle - *EMV2 Home*—>DBA Management Pack
—>Storage Manager
- 2** Enter the login information, and click OK.
- 3** Expand the Tablespaces folder and select the data file.
- 4** In the General page of the property sheet, update the file information.
- 5** Click Apply.

Note

- These commands verify that the file exists in the new location; they do not create or move files.
- Always provide complete filenames (including their paths) to identify the old and new data files.

Data Dictionary Information

Obtaining Tablespace Information

- **Tablespace information:**
 - **DBA_TABLESPACES**
 - **V\$TABLESPACE**
- **Data file information:**
 - **DBA_DATA_FILES**
 - **V\$DATAFILE**
- **Tempfile information:**
 - **DBA_TEMP_FILES**
 - **V\$TEMPFILE**

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Joining Data Dictionary Views

The performance views are built from information in the control file. To join V\$TABLESPACE with V\$DATAFILE or V\$TEMPFILE, join the views on the column TS# (tablespace number).

To join the data dictionary views on tablespaces with the views on data files, join the tables on the tablespace name.

Guidelines

Guidelines

- Use multiple tablespaces
- Specify default storage parameters for tablespaces
- Assign tablespace quotas to users
- Include directory path in filenames
- Use MINIMUM EXTENTS
- Use locally managed extents
- Can have 1023 data files per tablespace

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Use Multiple Tablespaces

Use multiple tablespaces for more flexibility when performing database operations.

- Separate user data from data dictionary data.
- Separate data segments from index segments
- Separate application data from each other.
- Store the data files of different tablespaces on separate disk drives to reduce I/O contention.
- Separate the rollback segment from data segments, to prevent a single disk failure from causing permanent loss of data.
- Take individual tablespaces offline while others remain online.
- Reserve tablespaces for a particular type of database use, such as high-update activity, read-only activity, or temporary segment storage.
- Back up individual tablespaces.

Specify Storage Parameters for the Tablespace

Specify default storage parameters for a tablespace to account for the size of a typical object that will be created in the tablespace.

Assign Tablespace Quotas to Users

Assign tablespace quotas, as necessary, to database users.

Include the Directory Path in Filenames

If you create or rename data files, specify the full filenames; otherwise the Oracle server creates the data files in the default directory of the database server.

Use MINIMUM EXTENTS to Control Fragmentation

With the setting of the MINIMUM EXTENT option, the DBA controls the fragmentation in the tablespace. This option can only be specified for a tablespace, not for the storage of individual objects.

Limits

The maximum number of tablespaces per database is 64 KB.

The operating system–specific limit on the maximum number of data files allowed in a tablespace is typically 1023 files; however, this number varies by operating system.

Summary

Summary

In this lesson, you should have learned how to:

- Use tablespaces to separate data
- Resize tablespaces by:
 - Adding data files
 - Extending data files
- Use locally managed tablespaces
- Use temporary tablespaces

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Quick Reference

Context	Reference
Initialization parameters	DB_FILES
Dynamic performance views	V\$DATAFILE V\$TEMPFILE V\$TABLESPACE
Data dictionary views	DBA_DATA_FILES DBA_TABLESPACES DBA_TEMP_FILES
Commands	CREATE TABLESPACE ... DATAFILE ... DEFAULT STORAGE...MINIMUM EXTENT CREATE TEMPORARY TABLESPACE CREATE TABLESPACE ... DATAFILE AUTOEXTEND ALTER TABLESPACE ... ADD DATAFILE ... AUTOEXTEND ALTER DATABASE DATAFILE ... RESIZE ALTER TABLESPACE...DEFAULT STORAGE... MINIMUM EXTENT ALTER TABLESPACE ... RENAME DATAFILE... ALTER DATABASE RENAME FILE... ALTER TABLESPACE ... READ ONLY ALTER TABLESPACE ... READ WRITE ALTER TABLESPACE ... OFFLINE DROP TABLESPACE
Packaged procedures and functions	None

Storage Structure and Relationships

Objectives

Objectives

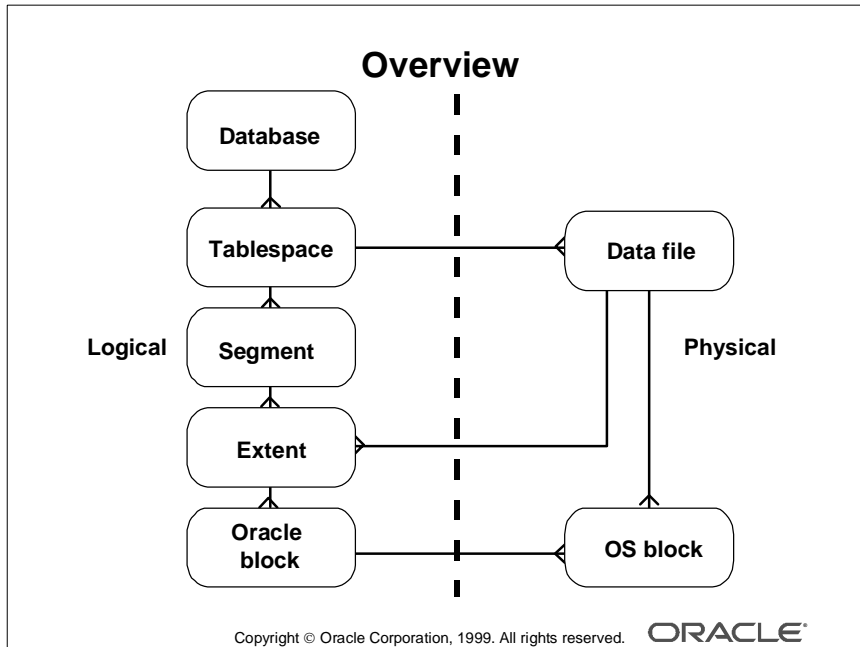
After completing this lesson, you should be able to do the following:

- **Describe the logical structure of the database**
- **List the segment types and their uses**
- **List the keywords that control block space usage**
- **Obtain information about storage structures from the data dictionary**
- **List the criteria for separating segments**

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Overview

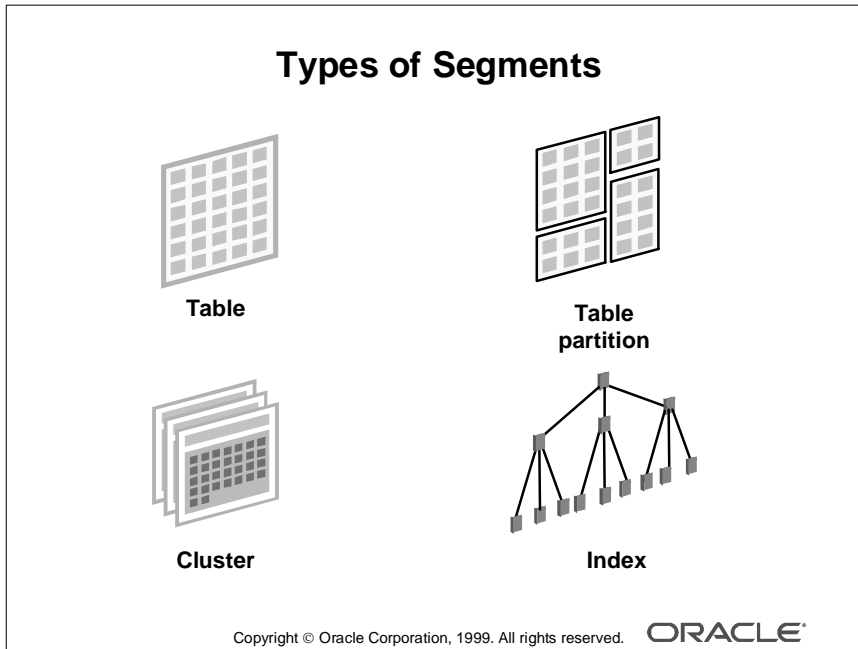


Database Architecture

The previous lesson discussed the storage structure of a database, its tablespaces, and its data files. This lesson continues the discussion of database storage by examining segments, extents, and data blocks.

Note: The commands and guidelines to manage the different types of segments are discussed in detail in the course *Oracle8 for Developers*.

Types of Segments



Segments

Segments are space-occupying objects in a database. They use space in the data files of a database. This section describes the different types of segments.

Table

A table is the most common means of storing data within a database. A table segment stores that data for a table that is neither clustered nor partitioned. Data within a table segment is stored in no particular order, and the database administrator has very little control over the location of rows within the blocks in a table. All the data in a table segment must be stored in one tablespace.

Table Partition

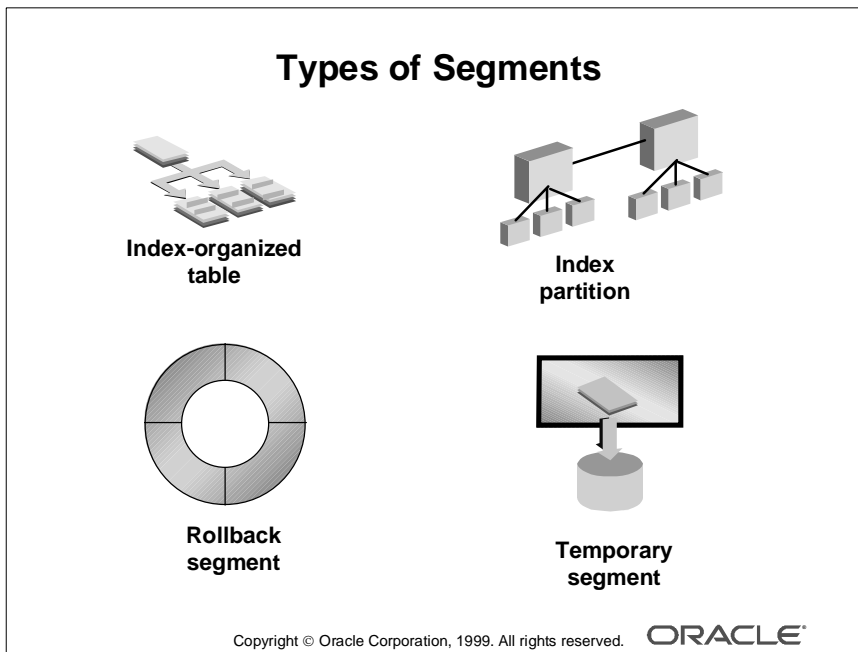
Scalability and availability are major concerns when there is a table in a database with high concurrent usage. In such cases, data within a table may be stored in several partitions, each of which resides in a different tablespace. The Oracle server currently supports partitioning by a range of key values or by a hashing algorithm. If a table is partitioned, each partition is a segment and storage parameters can be specified to control them independently. Use of this type of segment requires the Partitioning option within the Oracle8i Enterprise Edition.

Cluster

A cluster, like a table, is a type of data segment. Rows in a cluster are stored based on key column values. A cluster may contain one or more tables. Tables in a cluster belong to the same segment and share the same storage characteristics. The rows in a clustered table can be accessed with an index or hashing algorithm.

Index

All the entries for a particular index are stored within one index segment. If a table has three indexes, three index segments are used. The purpose of this segment is to look up the location of rows in a table based on a specified key.



Index-Organized Table

In an index-organized table, data is stored within the index based on the key value. An index-organized table does not need a table lookup, because all the data can be retrieved directly from the index tree.

Index Partition

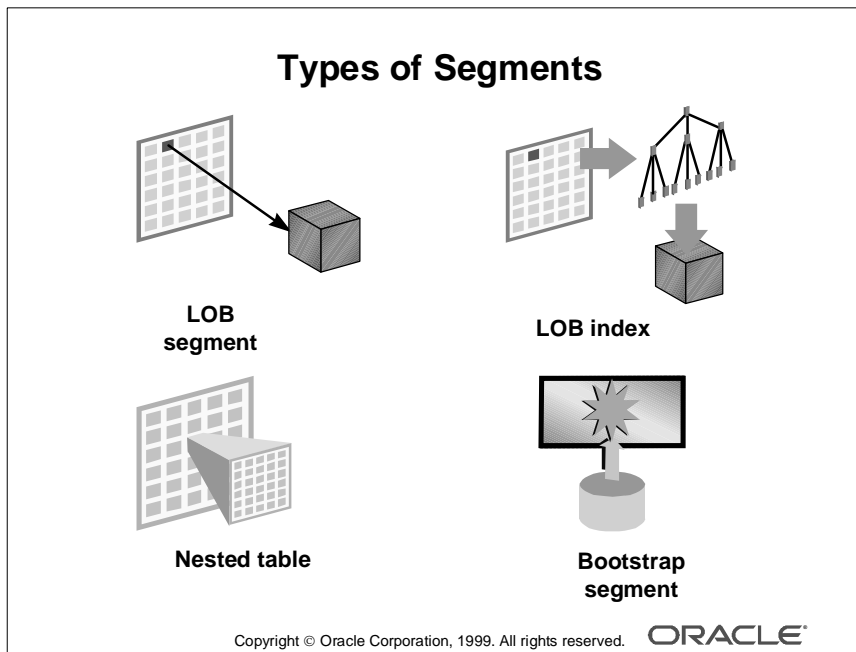
An index can be partitioned and spread across several tablespaces. In this case, each partition in the index corresponds to a segment and cannot span multiple tablespaces. The primary use of a partitioned index is to minimize contention by spreading index I/O. Use of this type of segment requires the Partitioning option within the Oracle8i Enterprise Edition.

Rollback Segment

A rollback segment is used by a transaction that is making changes to a database. Before changing the data or index blocks, the old value is stored in the rollback segment. This allows a user to undo changes made.

Temporary Segment

When a user executes commands such as CREATE INDEX, SELECT DISTINCT, and SELECT GROUP BY, the Oracle server tries to perform sorts in memory. When a sort needs more space than the space available in memory, intermediate results are written to the disk. Temporary segments are used to store these intermediate results.



LOB Segment

One or more columns in a table can be used to store large objects (LOBs) such as text documents, images, or videos. If the column is large, the Oracle server stores these values in separate segments known as LOB segments. The table only contains a locator or a pointer to the location of the corresponding LOB data.

LOB Index

An LOB index segment is created implicitly when an LOB segment is created. The LOB index is used to look up specific LOB column values.

Nested Table

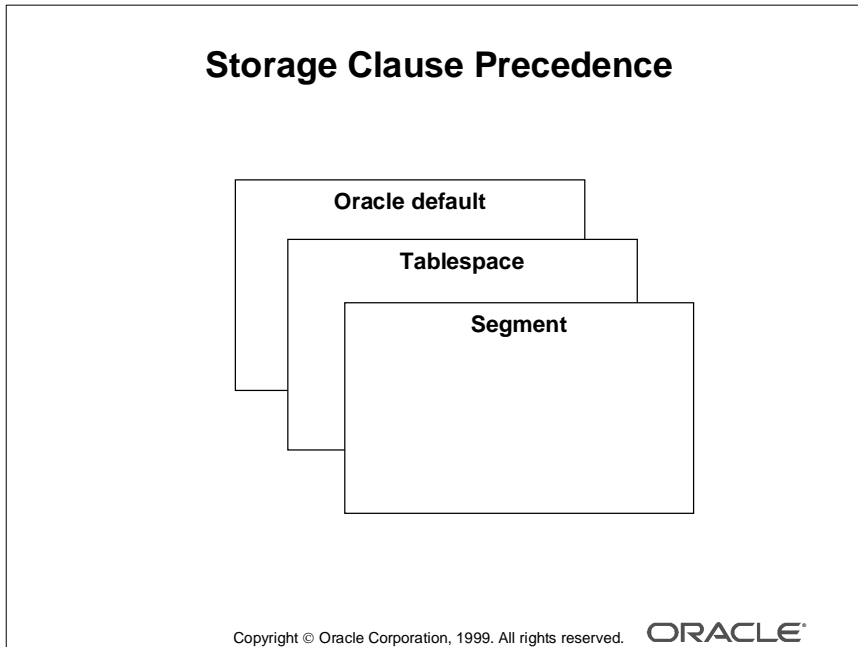
A column in a table may be made up of a user-defined table, as in the case of items within an order. In such cases, the inner table, which is known as a nested table, is stored as a separate segment. Using nested tables requires the Objects option of the Oracle8i Enterprise Edition.

Bootstrap Segment

A bootstrap segment, also known as a cache segment, is created by the `sql.bsq` script when a database is created. This segment helps to initialize the data dictionary cache when the database is opened by an instance.

The bootstrap segment cannot be queried or updated and does not require any maintenance by the database administrator.

Storage Clause Precedence



Storage Parameters

A storage clause can be specified at the segment level to control how extents are allocated to a segment.

- Any storage parameter specified at the segment level overrides the corresponding option set at the tablespace level, except for the **MINIMUM EXTENT** or **UNIFORM SIZE** tablespace parameter.
- When storage parameters are not set explicitly at the segment level, they default to those at the tablespace level.
- When storage parameters are not set explicitly at the tablespace level, the Oracle server system defaults are used.

Other Considerations

- If storage parameters are altered, the new options apply only to the extents not yet allocated.
- Some parameters cannot be specified at the tablespace level. These parameters need to be specified at the segment level only.
- If minimum extent size has been specified for the tablespace, this size will apply to all extents that are allocated for segments in the tablespace in the future.

Extent Allocation and Deallocation

Extent Allocation and Deallocation

- **Allocated when the segment is:**
 - Created
 - Extended
 - Altered
- **Deallocated when the segment is:**
 - Dropped
 - Altered
 - Truncated
 - Automatically resized (rollback segments only)

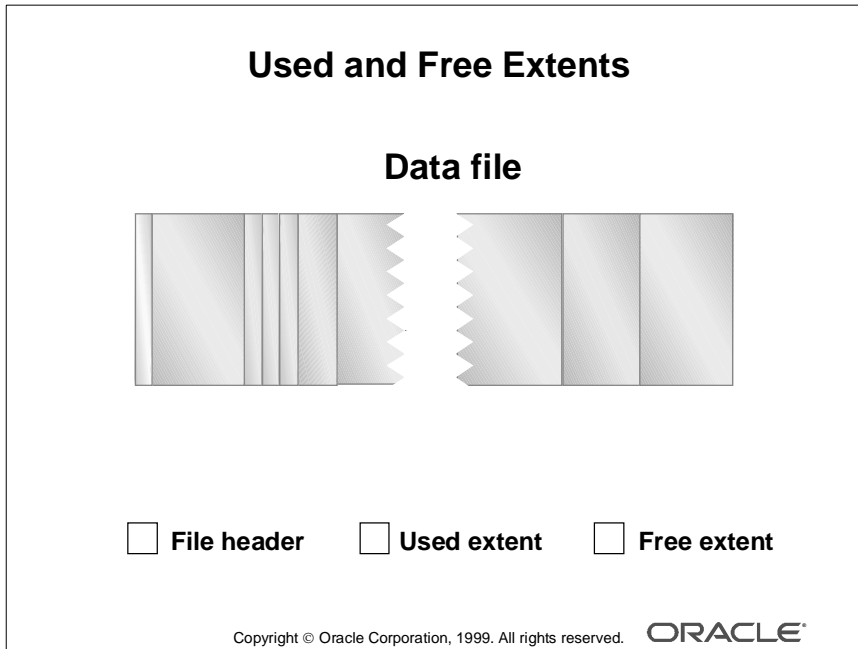
Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Resizing Rollback Segments

Resizing of rollback segments is discussed in detail in the lesson “Managing Rollback Segments.”

Used and Free Extents



Extents

When a tablespace is created, the data files in the tablespace contain the following elements:

- A header block, which is the first block in the file
- One free extent consisting of the remaining part of the data file

As segments are created, they are allocated space from the free extents in a tablespace. Contiguous space used by a segment is referred to as a *used extent*. When segments release space, the extents that are released are added to the pool of free extents available in the tablespace. Frequent allocation and deallocation of extents could lead to fragmentation of space within data files in a tablespace.

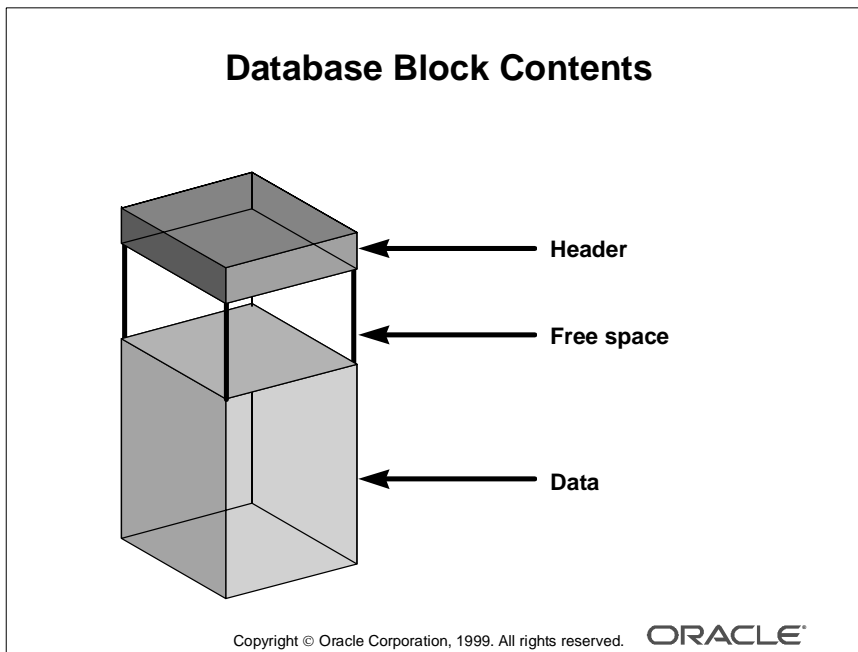
Using Block Space Utilization Parameters

Database Block: Review

- Minimum unit of I/O
- Consists of one or more OS blocks
- Set by DB_BLOCK_SIZE
- Set at database creation

Copyright © Oracle Corporation, 1999. All rights reserved.

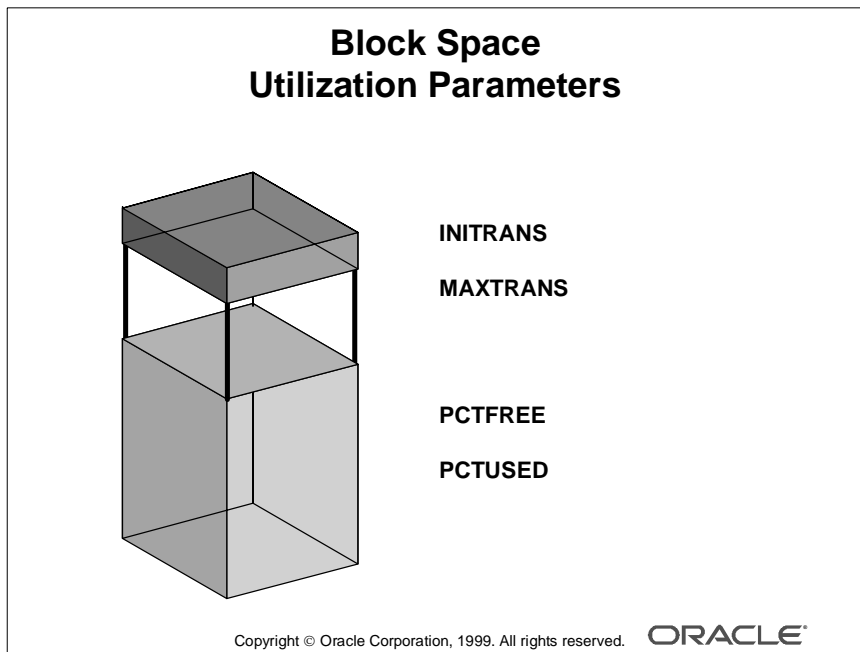
ORACLE®



Data Blocks

Oracle data blocks contain:

- **Block header:** The header contains the data block address, table directory, row directory, and transaction slots that are used when transactions make changes to rows in the block. Block headers grow from the top down.
- **Data space:** Row data is inserted into the block from the bottom up.
- **Free space:** The free space in a block is in the middle of the block, allowing both the header and the row data space to grow when necessary. The free space in a block is contiguous, initially. However, deletions and updates may fragment the free space in the block. The free space in the block is coalesced by the Oracle server when necessary.



Block Space Utilization Parameters

Block space utilization parameters can be used to control the use of space in data and index segments.

Parameters Controlling Concurrency

INITRANS and MAXTRANS specify the initial and the maximum number of transaction slots, which are created in an index or a data block. The transaction slots are used to store information about transactions that are making changes to the block at a point in time. A transaction only uses one transaction slot, even if it is changing more than one row or index entry.

INITRANS, which defaults to 1 for a data segment and 2 for an index segment, guarantees a minimum level of concurrency. For example, if set to 3, INITRANS ensures that at least three transactions can concurrently make changes to the block. If necessary, additional transaction slots can be allocated from the free space in the block to permit more concurrent transactions to modify rows in the block.

MAXTRANS, which has a default value of 255, sets the limit for the number of concurrent transactions that can make changes to a data or an index block. When set, this value restricts use of space for transaction slots and therefore guarantees that there is sufficient space in the block for use by row or index data.

Parameters Controlling the Use of Data Space

PCTFREE for a data segment specifies the percentage of space in each data block reserved for growth resulting from updates to rows in the block. The default for PCTFREE is 10 percent.

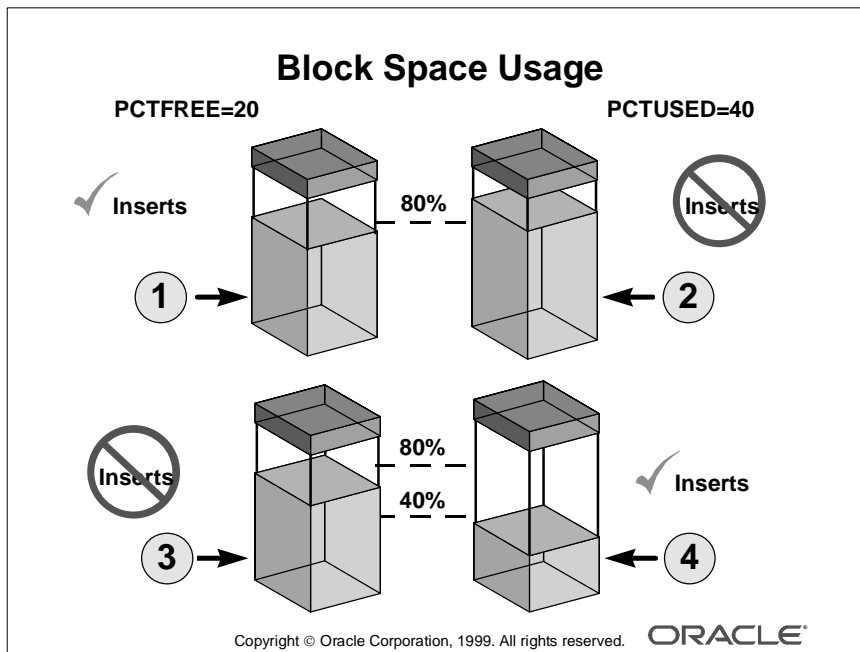
PCTUSED for a data segment represents the minimum percentage of used space that the Oracle server tries to maintain for each data block of the table. A block is put back on the free list when its used space falls below PCTUSED. The free list of a segment is a list of blocks that are candidates for accommodating future inserts. A segment, by default, is created with one free list. Segments can be created with a higher number of free lists by setting the FREELISTS parameter of the storage clause. The default for PCTUSED is 40 percent.

Both PCTFREE and PCTUSED are calculated as percentages of available data space; that is, the block space that remains after deducting the header space from the total block size.

Block space utilization parameters can only be specified for segments and cannot be set at the tablespace level.

Note: The use of these parameters for indexes is discussed in detail in the lesson “Managing Indexes.”

Specifying FREELISTS is discussed in detail in the course *Enterprise DBA Part 2: Performance and Tuning*.



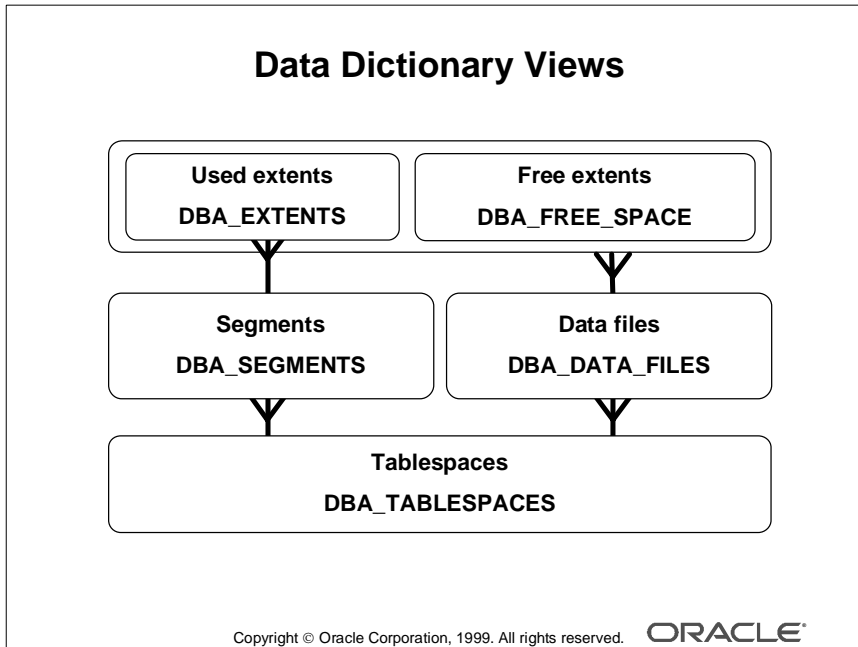
Block Space Usage

The following steps explain how space within a block is managed for a data segment with `PCTFREE=20` and `PCTUSED=40`:

- 1** Rows are inserted into the block until the free space in the block is equal to or less than 20%. The block is no longer available for inserts when rows occupy 80% ($100 - \text{PCTFREE}$) or more of the available data space in the block.
- 2** The remaining 20% can be used when the size of a row increases. For example, a column that was originally NULL is updated to be assigned a value. Thus, block utilization may be in excess of 80% as a result of updates.
- 3** If rows are deleted in the block or if rows decrease in size as a result of updates, block utilization may fall below 80%. However, a block is not used for inserts until the utilization falls below `PCTUSED`, which in this example is 40%.
- 4** When the utilization falls below `PCTUSED`, the block is available for inserts. As rows are inserted into the block, the utilization of the block increases and the cycle repeats starting with step 1.

Note: Guidelines for setting `PCTFREE` and `PCTUSED` are discussed in the lessons on tables and indexes, “Managing Tables” and “Managing Indexes,” respectively.

Obtaining Information About Storage Structures



Querying the Data Dictionary

The relationships between tablespaces, data files, segments, and free and used extents can be viewed by querying the data dictionary.

When a tablespace with one or more files is created, a row is added to `DBA_TABLESPACES`. For each file in the database, a row is added to `DBA_DATA_FILES`. At this stage, the space in each data file, excluding the file header, shows up as one free extent in `DBA_FREE_SPACE`.

When a segment is created, a row is visible in `DBA_SEGMENTS`. The space allocated to the extents in this segment can be viewed from `DBA_EXTENTS`, while `DBA_FREE_SPACE` is adjusted to show lower free space in the files where the extents have been created for the segment.

All the space in a file (excluding the header block) must be accounted for in either `DBA_FREE_SPACE` or `DBA_EXTENTS`.

Querying DBA_SEGMENTS

Querying DBA_SEGMENTS

- **General information**
 - OWNER
 - SEGMENT_NAME
 - SEGMENT_TYPE
 - TABLESPACE_NAME
- **Storage settings**
 - INITIAL_EXTENT
 - NEXT_EXTENT
 - MIN_EXTENTS
 - MAX_EXTENTS
 - PCT_INCREASE
- **Size**
 - EXTENTS
 - BLOCKS
 - BYTES
- **Other information**
 - Location
 - Tuning

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

DBA_SEGMENTS

Query the DBA_SEGMENTS view to get the number of extents and blocks allocated to a segment.

```
SQL> SELECT segment_name,tablespace_name,extents,blocks
2  FROM dba_segments
3  WHERE owner='SCOTT' ;
```

SEGMENT_NAME	TABLESPACE_NAME	EXTENTS	BLOCKS
-----	-----	-----	-----
EMP	DATA01	5	55
DEPT	DATA01	1	5
BONUS	DATA01	1	5
SALGRADE	DATA01	1	5

5 rows selected.

Querying DBA_EXTENTS

Querying DBA_EXTENTS

- Identification
 - OWNER
 - SEGMENT_NAME
 - EXTENT_ID
- Size
 - BLOCKS
 - BYTES
- Location
 - TABLESPACE_NAME
 - RELATIVE_FNO
 - FILE_ID
 - BLOCK_ID

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

DBA_EXTENTS

Use the DBA_EXTENTS view to check the extents for a given segment.

```
SQL> SELECT extent_id,file_id,block_id,blocks
2  FROM dba_extents
3  WHERE owner='SCOTT'
4  AND segment_name='EMP' ;
```

EXTENT_ID	FILE_ID	BLOCK_ID	BLOCKS
-----	-----	-----	-----
0	4	2	5
1	4	27	5
2	4	32	10
3	4	42	15
4	4	57	20

5 rows selected.

Querying DBA_FREE_SPACE

Querying DBA_FREE_SPACE

- **Location**
 - TABLESPACE_NAME
 - RELATIVE_FNO
 - FILE_ID
 - BLOCK_ID
- **Size**
 - BYTES
 - BLOCKS

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

DBA_FREE_SPACE

Use the DBA_FREE_SPACE view to check the extents for a given segment.

```
SQL> SELECT tablespace_name, count(*),
2         max(blocks), sum(blocks)
3 FROM dba_free_space
4 GROUP BY tablespace_name;
```

TABLESPACE_NAME	COUNT(*)	MAX(BLOCKS	SUM(BLOCKS
-----	-----	-----	-----
DATA01	2	1284	1533
RBS	3	2329	2419
SORT	1	1023	1023
SYSTEM	1	5626	5626
TEMP	1	2431	2431

5 rows selected.

Planning the Location of Segments

Organizing Tablespaces Based on Fragmentation Propensity

Tablespace	Usage	Fragmentation
SYSTEM	Data dictionary	Zero
TOOLS	Applications	Very low
DATA _n	Data segments	Low
INDEX _n	Index segments	Low
RBS _n	Rollback segments	High
TEMP _n	Temporary segments	Very high*

* Relevant only if tablespace PERMANENT

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Fragmentation

Different types of segments have varying propensities for fragmentation. It is recommended that segments be placed in different tablespaces in order to minimize the waste of space.

Types of Objects and Fragmentation

The recommended structure for the tablespaces, their uses, and their fragmentation propensities are shown in the table. The following are the different types of objects listed in increasing order of fragmentation propensity:

- Data dictionary objects, with the exception of the audit table, are never dropped or truncated, and therefore are not likely to fragment the tablespace.
- Space used for repositories of applications such as Oracle Enterprise Manager and Oracle Designer are only deallocated while reorganizing these structures. Since these tables are seldom reorganized, these objects have a very low propensity for fragmentation.
- Data and index segments used for user-written applications may need to be reorganized more frequently than repositories. They have a higher propensity for fragmentation than application repositories.

Types of Objects and Fragmentation (continued)

- Because rollback segments can deallocate extents automatically, they are likely to cause fragmentation in a system with high update activity.
- Temporary segments in permanent tablespaces can release space quite frequently, and therefore they must be located in separate tablespaces.

Influence of Segment Life Span

Segments may have different life spans in a database. For example, in a project-based environment, such as in a software development company, all data relating to an application may need to be purged when the development of the system is complete. Using a separate set of tablespaces for locating these segments may help during cleanup. At the end of the project, the entire tablespace can be backed up and dropped to provide space for other applications using the database.

Other Criteria for Organizing Tablespaces

The DBA may also want to separate tablespaces to:

- Control space allocation and assign space usage limits to users
- Control availability of data by taking individual tablespaces online or offline
- Distribute data storage across devices to improve I/O performance and to reduce contention against a single disk
- Perform partial backup and partial recovery operations
- Keep large amounts of static data on read-only devices

Summary

Summary

In this lesson, you should have learned how to:

- Use tablespaces to:
 - Separate segments to ease administration
 - Control users' space allocation
- Categorize segments by the type of information stored in the segment
- Determine extent sizes using the storage clause
- Control block space utilization

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Quick Reference

Context	Reference
Initialization parameters	DB_BLOCK_SIZE
Dynamic performance views	None
Data dictionary views	DBA_TABLESPACES DBA_DATA_FILES DBA_SEGMENTS DBA_EXTENTS DBA_FREE_SPACE DBA_FREE_SPACE_COALESCED

10

Managing Rollback Segments

Objectives

Objectives

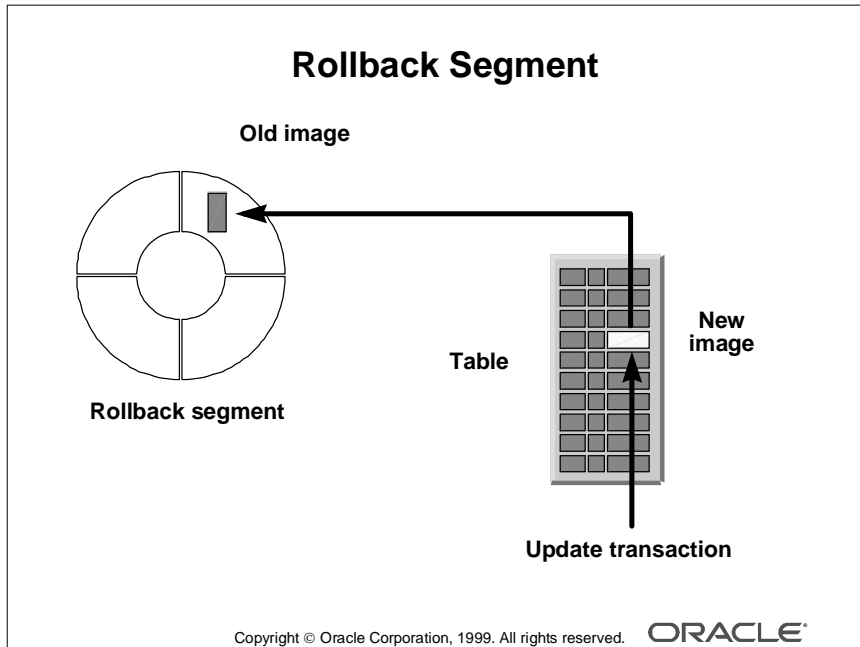
After completing this lesson, you should be able to do the following:

- **Create rollback segments using appropriate storage settings**
- **Maintain rollback segments**
- **Plan the number and size of rollback segments**
- **Obtain rollback segment information from the data dictionary**
- **Troubleshoot common rollback segment problems**

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Overview



Lesson Overview

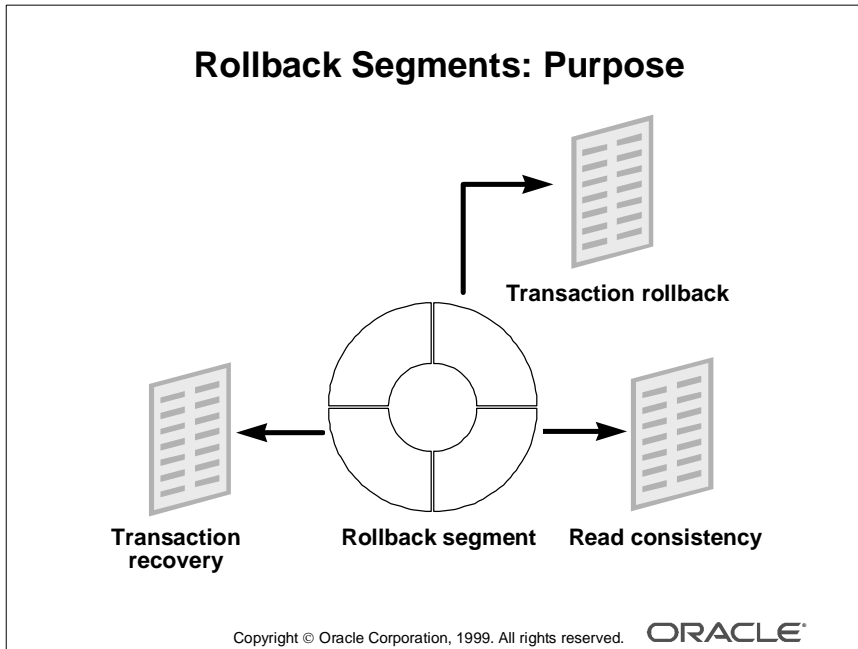
A rollback segment is used to save the old value when a process changes data in a database. It stores the location of the data and the data as it existed before being modified.

The header of a rollback segment contains a transaction table where information about the current transactions using the rollback segment is stored.

A transaction can use only one rollback segment to store all of its rollback (undo) records.

Many concurrent transactions can write to one rollback segment.

Rollback Segments



Transaction Rollback

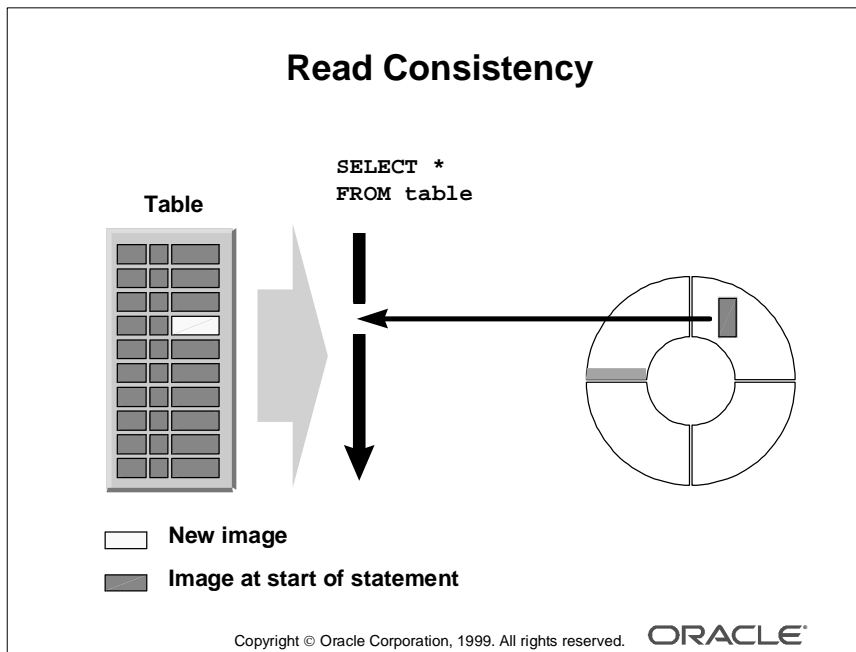
When a transaction changes a row in a table, the old image is saved in the rollback segment. If the transaction is rolled back, the Oracle server restores the original value by writing the value in the rollback segment back to the row.

Transaction Recovery

If the instance fails while transactions are in progress, the Oracle server needs to roll back any uncommitted changes when the database is opened again. This rollback is part of transaction recovery. Recovery is only possible because changes made to the rollback segment are also protected by the redo log files.

Read Consistency

While transactions are in progress, other users in the database should not see any uncommitted changes made by these transactions. In addition, a statement should not see any changes that are committed after the statement begins execution. The old values in the rollback segments are also used to provide the readers a consistent image for a given statement.



Read Consistency (continued)

The Oracle server guarantees that a statement sees data from a consistent time, even if that data is modified by other transactions.

When the Oracle server begins executing a SELECT statement, it determines the current system change number (SCN) and ensures that any changes not committed before this SCN are not processed by the statement. Consider the case where a long-running query is executed at a time when several changes are being made. If a block has changes that were not committed at the start of the query, the Oracle server constructs a read-consistent image of the block by retrieving the before-image of the changes from the rollback segment and applying the changes to a copy of the block in memory.

Transaction Read Consistency

Read consistency is always provided for a SQL statement. However, you can request read consistency for a read-only transaction by issuing the following command at the beginning of the transaction:

```
SET TRANSACTION READ ONLY;
```

Or, you can request read consistency for a transaction performing DML by issuing the following command at the beginning of the transaction:

```
SET TRANSACTION SERIALIZABLE;
```

In either case, the Oracle server provides data that is read consistent from the start of the transaction. Using SERIALIZABLE can have a negative impact on performance.

Types of Rollback Segments

- **SYSTEM:** Used for objects in the **SYSTEM** tablespace
- **Non-SYSTEM:** Used for objects in other tablespaces
 - **Private:** Acquired by a single instance
 - **Public:** Acquired by any instance
- **Deferred:** Used when tablespaces are taken offline with the **immediate** option

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

SYSTEM Rollback Segment

The **SYSTEM** rollback segment is created in the **SYSTEM** tablespace when a database is created. This rollback segment can only be used for changes made to objects in the **SYSTEM** tablespace.

Non-SYSTEM Rollback Segments

A database that has multiple tablespaces needs at least one non-**SYSTEM** rollback segment. A non-**SYSTEM** rollback segment, which is created by the database administrator, can be used for changes made to objects in any non-**SYSTEM** tablespace. There are two types of non-**SYSTEM** rollback segments.

Private Private rollback segments are segments that are brought online by an instance because they are listed in the parameter file. However, they can be brought online explicitly by issuing an **ALTER ROLLBACK SEGMENT** command.

Non-SYSTEM Rollback Segments (continued)

Public Public rollback segments form a pool of rollback segments available in a database. Public rollback segments are normally used with the Oracle Parallel Server to create a pool of rollback segments that can be used by any of the Parallel Server instances.

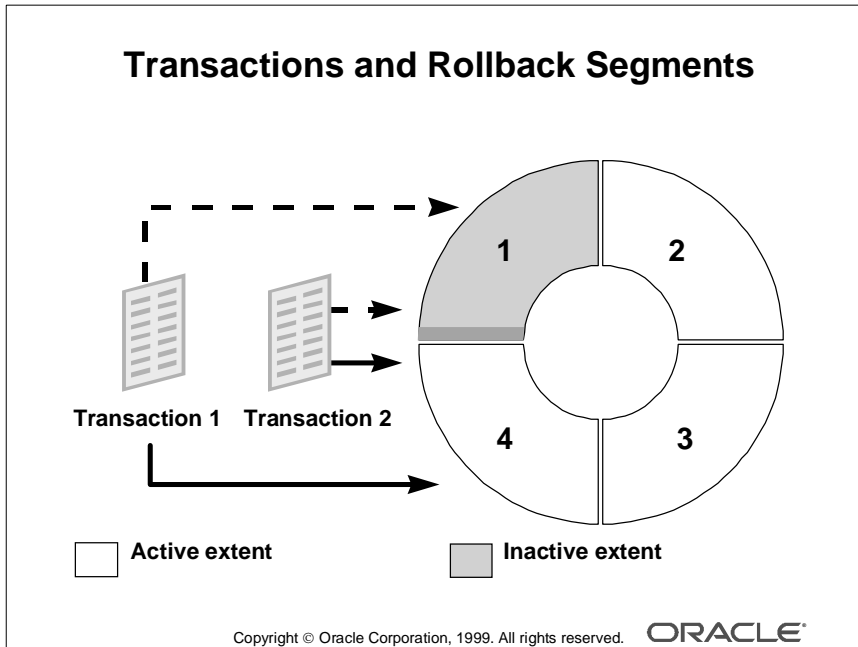
Note: The use of public rollback segments is discussed in the *Oracle8i Parallel Server Concepts and Administration* manual.

Deferred Rollback Segments

Deferred rollback segments may be created when a tablespace is brought offline. They are used to roll back transactions when the tablespace is brought back online. They are dropped automatically when they are no longer needed.

Because deferred rollback segments are maintained by the Oracle server, no maintenance is required on the part of the DBA.

Using Rollback Segments with Transactions



Allocation of a Rollback Segment

When a transaction begins, a rollback segment needs to be assigned to this transaction. A transaction may request a specific rollback segment using the following command:

```
SET TRANSACTION USE ROLLBACK SEGMENT rollback_segment
```

If no such request is made, the Oracle server chooses the rollback segment with the fewest transactions, and assigns it to the transaction.

Using Extents

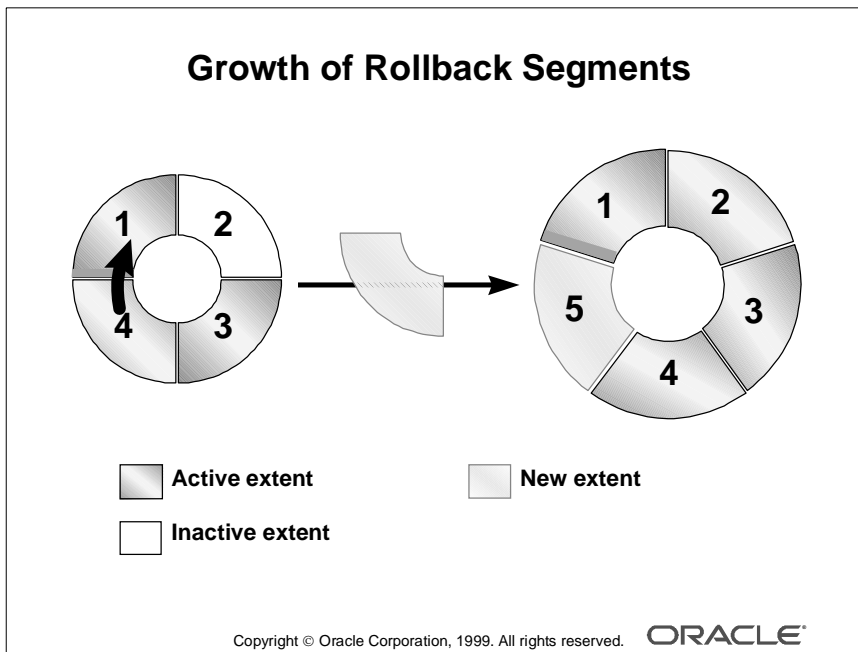
Transactions use extents of a rollback segment in a sequential, circular fashion, moving from one to the next after the current extent is full. A transaction writes an entry to its current location in the rollback segment and advances the current pointer by the size of the entry.

More than one transaction can write to the same extent of a rollback segment; however, each rollback segment block contains information from one and only one transaction.

Example

In the example in the slide, two transactions have been assigned to a rollback segment, which has four extents.

- 1 When the transactions commence, they begin writing to Extent 3 of the rollback segment.
- 2 As the two transactions generate more rollback information, they continue to write into Extent 3.
- 3 When Extent 3 is full, the transactions write to the next extent in the ring, which is Extent 4. When transactions start writing to a new extent as in this step, it is called a *wrap*.
- 4 When the last extent for the rollback segment (Extent 4) is full, the transactions can use the first in the ring (Extent 1) if it is free or inactive. An extent is only free or inactive if there are currently no active transactions using the extent—that is, all transactions that wrote to the extent have completed.

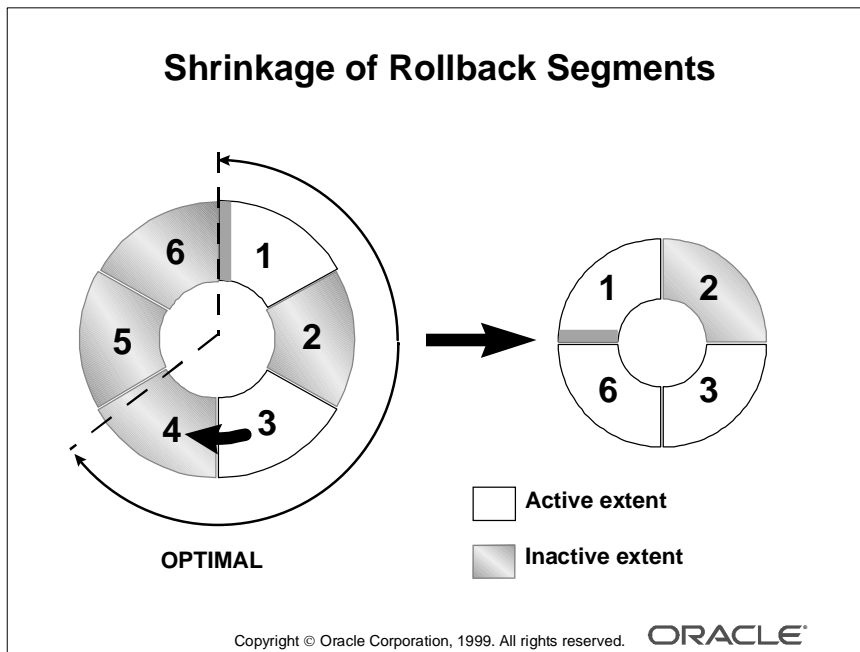


Growth of Rollback Segments

The pointer or the head of the rollback segment moves to the next extent when all blocks of the current extent are used and a transaction needs another block for more space. When the last extent is full, the pointer moves to the front of the first extent.

The pointer can only move to the next extent if that extent has no active transactions. The pointer cannot skip over an extent. If the next extent is being used, the transaction will allocate an additional extent for the rollback segment. This is called an *extend*.

A rollback segment may grow in this manner until it reaches the maximum number of extents specified by the MAXEXTENTS parameter.



OPTIMAL Parameter

The OPTIMAL parameter specifies the size in bytes that a rollback segment must shrink to, if possible. Specifying OPTIMAL minimizes the waste of space in a rollback segment. If the OPTIMAL parameter is specified, a rollback segment can release space on completion of transactions that caused the growth.

The deallocation of extents is not done as soon as transactions are completed. The process of deallocating extents is performed only when the head moves from one extent to the next. Extents are deallocated if both of these conditions are true:

- The current size of the rollback segment exceeds OPTIMAL
- There are contiguous inactive extents

The Oracle server tries to deallocate the size of the rollback segment until it is equal to OPTIMAL, but may have to stop short if the next extent to be deallocated is in use.

The Oracle server always deallocates the oldest inactive extents, as they are least likely to be used for read consistency.

Creating Rollback Segments

Creating Rollback Segments

```
CREATE ROLLBACK SEGMENT rbs01
TABLESPACE rbs
STORAGE (
    INITIAL      100K
    NEXT        100K
    MINEXTENTS  20
    MAXEXTENTS  100
    OPTIMAL     2000K );
```

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Syntax

Use the following command to create a rollback segment:

```
CREATE [PUBLIC] ROLLBACK SEGMENT rollback_segment
[TABLESPACE tablespace]
[STORAGE ([INITIAL integer[K|M]]
          [NEXT integer[K|M]]
          [MINEXTENTS integer]
          [MAXEXTENTS {integer|UNLIMITED}]
          [OPTIMAL {integer[K|M]|NULL}]
        )
]
```


Restrictions

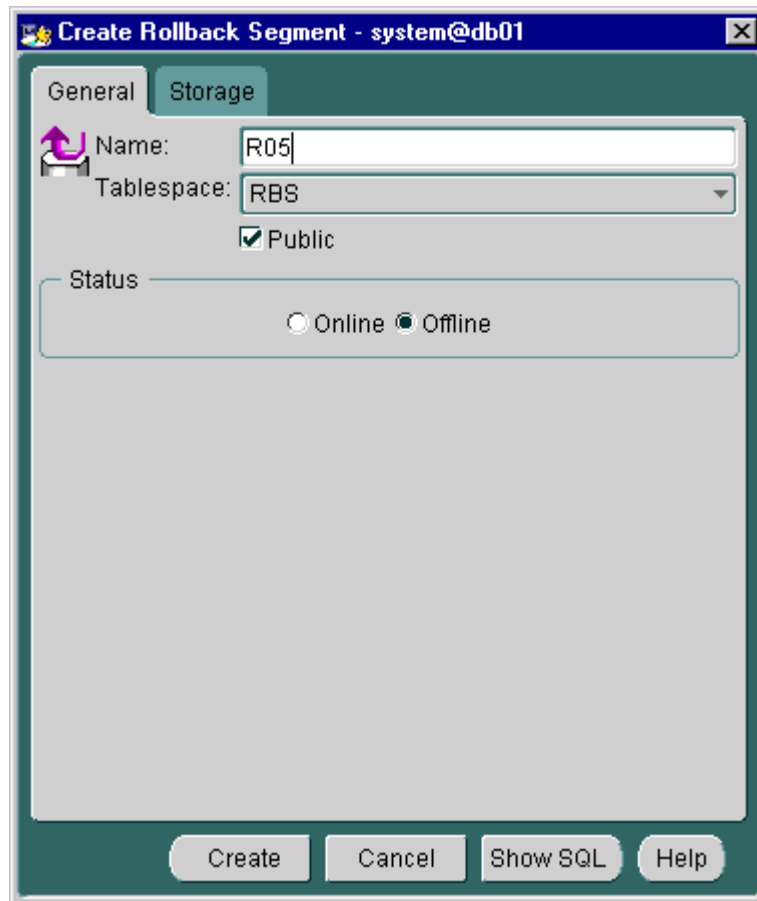
- A rollback segment can be specified as either **PUBLIC** or **PRIVATE** (the default) at the time of creation and cannot be changed.
- For a rollback segment, **MINEXTENTS** must be at least two.
- **PCTINCREASE** cannot be specified for a rollback segment and is always set to 0.
- **OPTIMAL**, if specified, must be at least equal to the initial size of the rollback segment, which is the space used by the number of extents defined by **MINEXTENTS**.

Guidelines

- Always use **INITIAL = NEXT** for rollback segments to ensure that all extents are of the same size.
- Set the **OPTIMAL** value to minimize the allocation and deallocation of rollback segment extents.
- Avoid setting **MAXEXTENTS** to **UNLIMITED**. This could cause unnecessary extension of a rollback segment and possibly of data files due to a program error.
- Always place rollback segments in a separate, exclusive tablespace to minimize contention and fragmentation.

How to Use Oracle Enterprise Manager to Create a New Rollback Segment

- 1 Launch Storage Manager and connect directly to the database:
Start—>Programs—>Oracle - *EMV2 Home*—>DBA Management Pack
—>Storage Manager
- 2 Enter the login information, and click OK.
- 3 Select the Rollback Segments folder, and choose Create from the right mouse menu.
- 4 In the General page of the property sheet, enter the name, tablespace, and type. Optionally, select the Online option button.



- 5 In the Extents page of the property sheet, enter storage information.
- 6 Click Create.

Use Rollback—>Create Like to create a new rollback segment that uses the same tablespace and settings as an existing rollback segment.

Bringing Rollback Segments Online

- Use the following command to make a rollback segment available:

```
ALTER ROLLBACK SEGMENT rbs01 ONLINE;
```

- Specify the following initialization parameter to ensure rollback segments are brought online at startup:

```
ROLLBACK_SEGMENTS=(rbs01, rbs02)
```

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

The ALTER ROLLBACK SEGMENT Command

When a rollback segment is created, it is offline and cannot be used. To make the rollback segment available for use by transactions, use the ALTER ROLLBACK SEGMENT command and bring it online.

Syntax

Use the following command to make a rollback segment available:

```
ALTER ROLLBACK SEGMENT rollback_segment ONLINE;
```

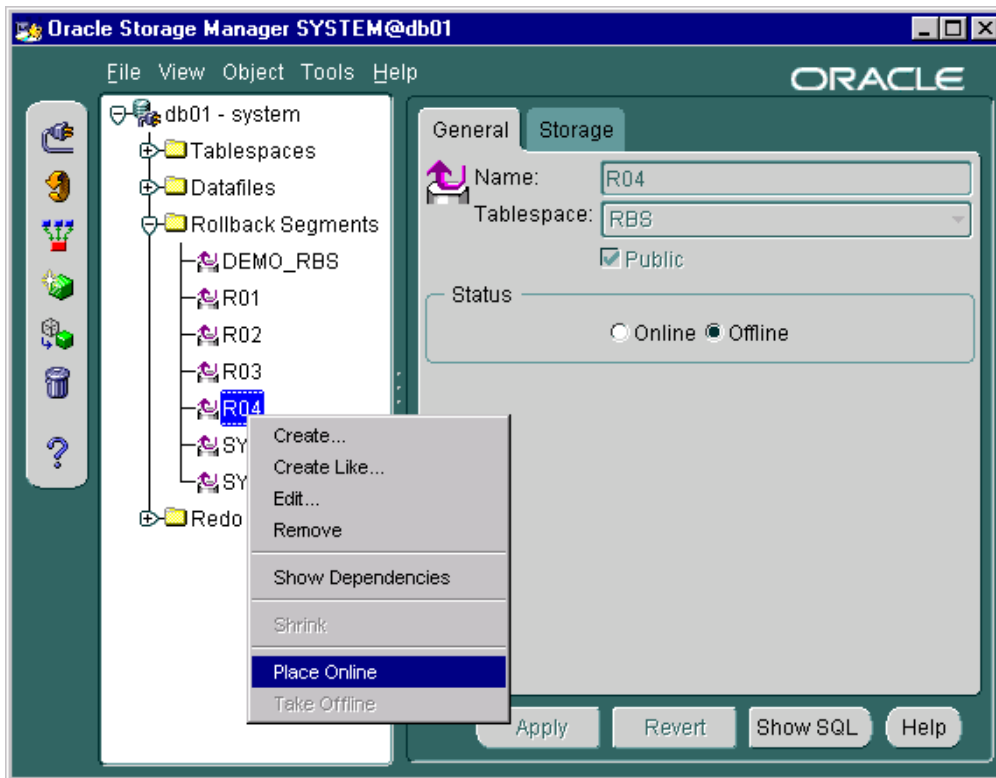
The number of rollback segments that can be brought online by an instance is limited by the MAX_ROLLBACK_SEGMENTS parameter. Set this value to one more than the number of non-SYSTEM rollback segments required for the instance.

A rollback segment is only online until the instance is shut down. To ensure that a rollback segment is always brought online by an instance, specify the name of the rollback segment in the parameter file as shown in the example below:

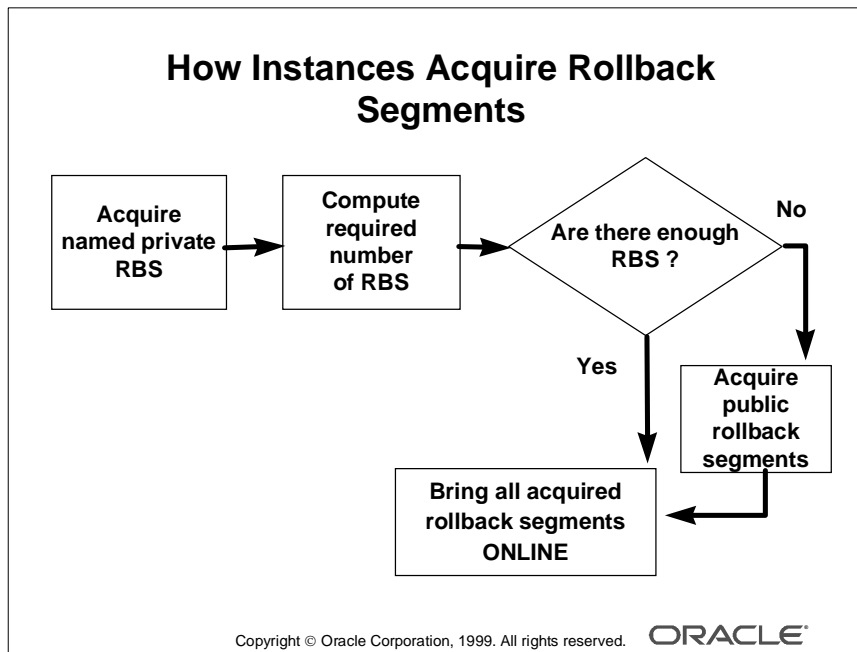
```
ROLLBACK_SEGMENTS=(rbs01, rbs02)
```

How to Use Oracle Enterprise Manager to Bring a Rollback Segment Online

- 1 Launch Storage Manager and connect directly to the database:
Start—>Programs—>Oracle - *EMV2 Home*—>DBA Management Pack
—>Storage Manager
- 2 Enter the login information, and click OK.
- 3 Expand the Rollback Segments folder.
- 4 Select the rollback segment.
- 5 Select Place Online from the right mouse menu.



- 6 In the dialog box, click Yes.



How Instances Acquire Rollback Segments

The following steps explain how rollback segments are acquired by an instance when it opens a database:

- 1 The instance acquires all rollback segments that are named in the initialization parameter `ROLLBACK_SEGMENTS`.
- 2 The `TRANSACTIONS` `init.ora` parameter is divided by the `TRANSACTIONS_PER_ROLLBACK_SEGMENT` `init.ora` parameter, and the result is the number of rollback segments needed by the instance. If this value is greater than the non-SYSTEM rollback segments already brought online by the instance, then the instance will acquire additional public rollback segments to make up for the shortfall. If there are insufficient public rollback segments, the database will be opened and available to users. No errors will be generated.

Maintaining Rollback Segments

Changing Rollback Segment Storage Settings

- Use the **ALTER ROLLBACK SEGMENT** command
- Can change **OPTIMAL** or **MAXEXTENTS**

```
ALTER ROLLBACK SEGMENT rbs01  
STORAGE( MAXEXTENTS 200 );
```

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Syntax

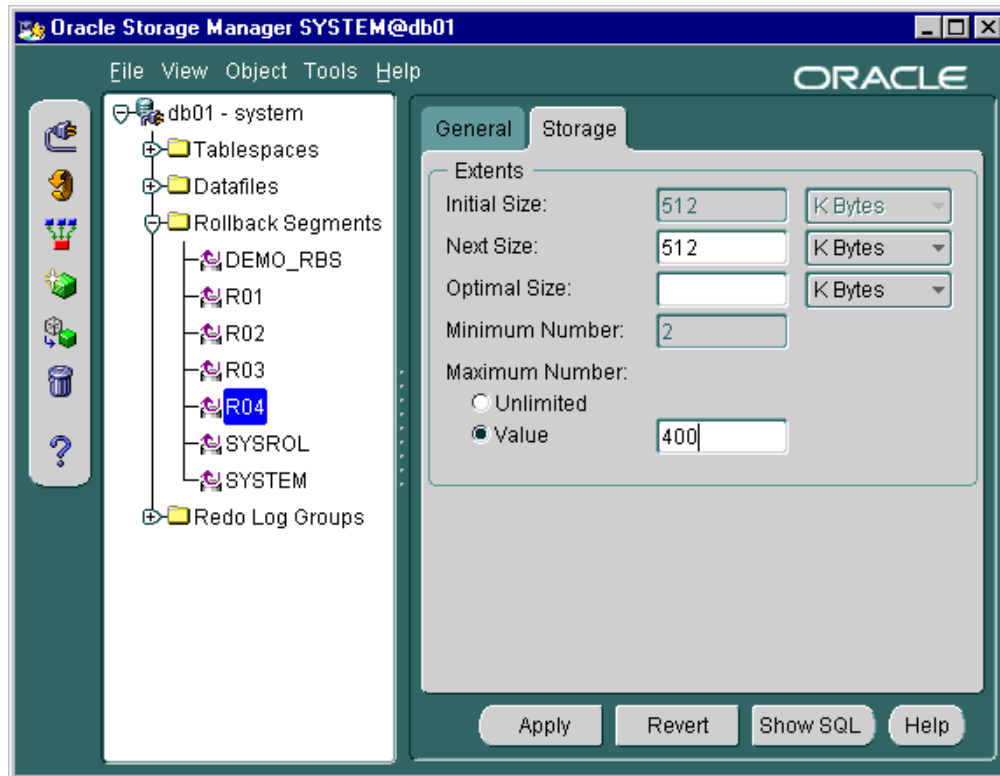
The storage parameters for a rollback segment can be changed using the **ALTER ROLLBACK SEGMENT** command.

```
ALTER ROLLBACK SEGMENT rollback_segment  
[STORAGE ( [NEXT integer[K|M]]  
            [MINEXTENTS integer]  
            [MAXEXTENTS {integer|UNLIMITED}]  
            [OPTIMAL {integer[K|M]|NULL}]  
          )  
]
```

Use this command to redefine the **OPTIMAL** or **MAXEXTENTS** parameters.

How to Use Oracle Enterprise Manager to Change Storage Settings

- 1 Launch Storage Manager and connect directly to the database:
Start—>Programs—>Oracle - *EMV2 Home*—>DBA Management Pack
—>Storage Manager
- 2 Enter the login information, and click OK.
- 3 Expand the Rollback Segments folder.
- 4 Select the rollback segment.
- 5 Select the Storage tab of the property sheet, and change the parameters.



- 6 Click Apply.

Deallocating Space from Rollback Segments

- Use the **ALTER ROLLBACK SEGMENT** command
- If extents are active, may not shrink to the requested size

```
ALTER ROLLBACK SEGMENT rbs01  
SHRINK TO 4M;
```

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

Deallocating Space from Rollback Segments

If **OPTIMAL** has been specified for a rollback segment, the Oracle server will attempt to deallocate extents to release space above the optimal size.

Manual Deallocation

To manually deallocate space from a rollback segment, use the following command:

```
ALTER ROLLBACK SEGMENT rollback_segment  
SHRINK [ TO integer [ K|M ]];
```

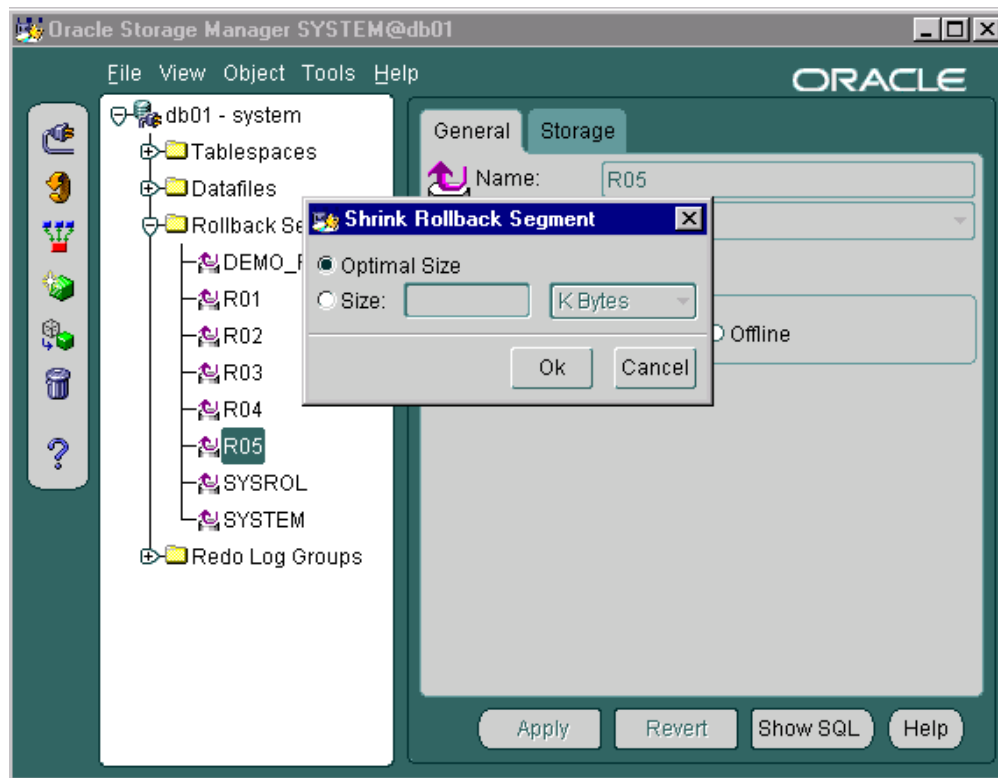
This command attempts to reduce the size of the rollback segment to the specified size, but will stop short if an extent cannot be deallocated because it is active.

If *integer* is not specified, the Oracle server attempts to deallocate extents until the size of the rollback segment is equal to **OPTIMAL**.

If the integer specified is larger than the current size of the rollback segment, this command is ignored.

How to Use Oracle Enterprise Manager to Shrink a Rollback Segment

- 1 Launch Storage Manager and connect directly to the database:
Start—>Programs—>Oracle - *EMV2 Home*—>DBA Management Pack
—>Storage Manager
- 2 Enter the login information, and click OK.
- 3 Expand the Rollback Segments folder.
- 4 Select the rollback segment.
- 5 Select Shrink from the right mouse button menu.
- 6 In the Shrink Rollback Segment dialog box, select Optimal Size to shrink the rollback segment to its optimal size. Alternatively, select Size and enter the size in kilobytes or megabytes.



- 7 Click OK.

Taking a Rollback Segment Offline

- Take a rollback segment offline to make it unavailable.
- If transactions are using the rollback segment, the status is temporarily changed to **PENDING OFFLINE**.

```
ALTER ROLLBACK SEGMENT rbs01  
OFFLINE;
```

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

Taking a Rollback Segment Offline

Take a rollback segment offline:

- To prevent new transactions from using a rollback segment
- If the rollback segment needs to be dropped

Syntax

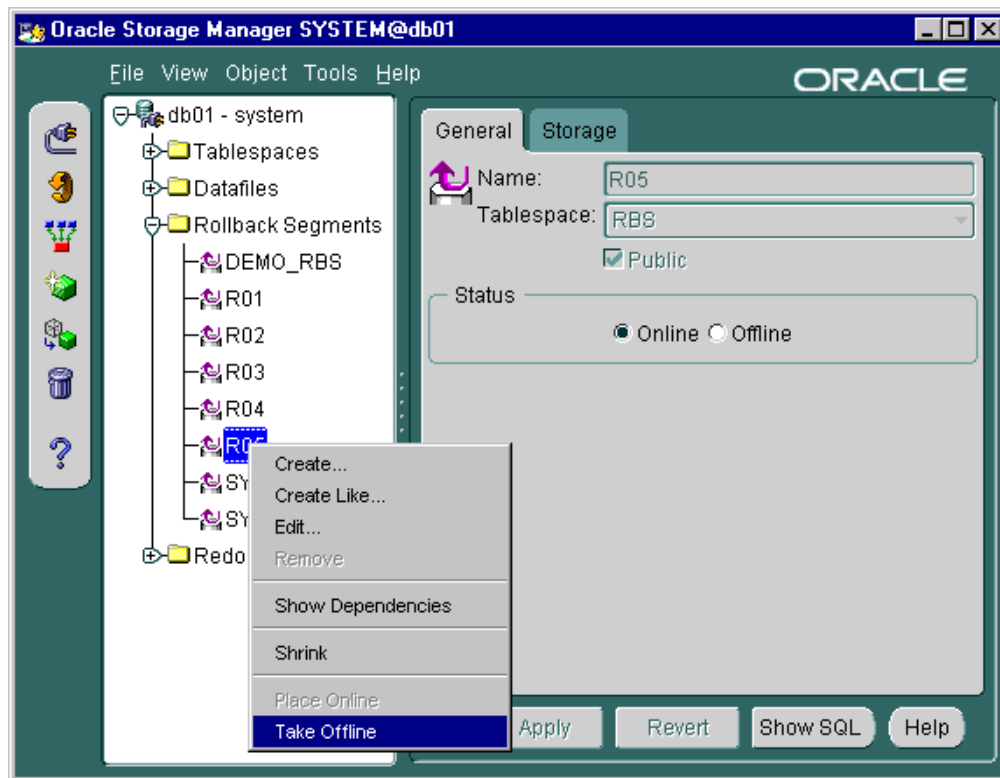
Use the following command to take a rollback segment offline:

```
ALTER ROLLBACK SEGMENT rollback_segment OFFLINE;
```

If there are transactions using the rollback segment at the time this statement is executed, the status of the rollback segment is set to **PENDING OFFLINE**, as seen from the dynamic performance view **V\$ROLLSTAT**. As soon as all existing transactions complete, the segment will be taken offline.

How to Use Oracle Enterprise Manager to Shrink a Rollback Segment

- 1 Launch Storage Manager and connect directly to the database:
Start—>Programs—>Oracle - *EMV2 Home*—>DBA Management Pack
—>Storage Manager
- 2 Enter the login information, and click OK.
- 3 Expand the Rollback Segments folder.
- 4 Select the rollback segment.
- 5 Select Take Offline from the right mouse button menu.



- 6 In the dialog box, click Yes.

Dropping Rollback Segments

A rollback segment must be offline before it can be dropped.

```
DROP ROLLBACK SEGMENT rbs01;
```

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

Syntax

Use the following command to drop a rollback segment:

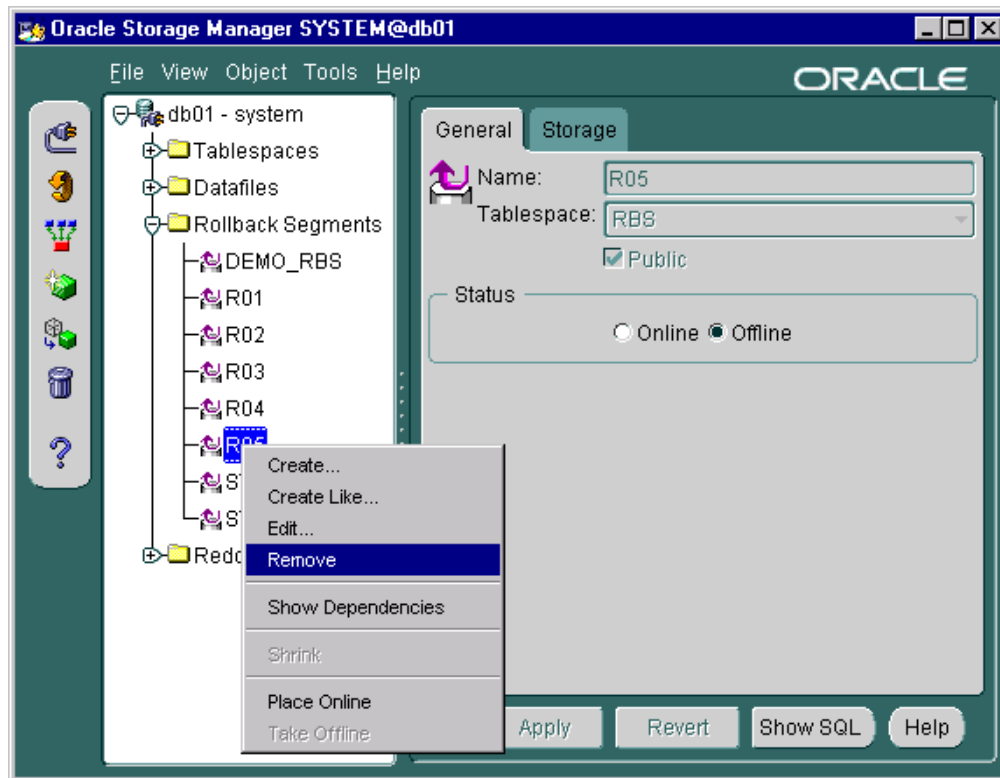
```
DROP ROLLBACK SEGMENT rollback_segment;
```

A rollback segment may need to be dropped if it is no longer needed or if it needs to be re-created with different storage settings for INITIAL, NEXT, or MINEXTENTS.

A rollback segment must be offline before it can be dropped.

How to Use Oracle Enterprise Manager to Drop a Rollback Segment

- 1 Launch Storage Manager and connect directly to the database:
Start—>Programs—>Oracle - *EMV2 Home*—>DBA Management Pack
—>Storage Manager
- 2 Enter the login information, and click OK.
- 3 Expand the Rollback Segments folder.
- 4 Select the rollback segment.
- 5 Select Remove from the right mouse button menu. The remove option will be enabled only if the rollback segment is offline.



- 6 In the dialog box, click Yes.

Obtaining Rollback Segment Information

Rollback Segments in the Database

DBA_ROLLBACK_SEGS

- **Identification:**
 - **SEGMENT_ID**
 - **SEGMENT_NAME**
- **Location: TABLESPACE_NAME**
- **Type: OWNER (PUBLIC or SYS)**
- **Status: STATUS (ONLINE or OFFLINE)**

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Obtaining Rollback Segment Information from the Data Dictionary

To obtain information about all the rollback segments in the database, query the DBA_ROLLBACK_SEGS view.

```
SQL> SELECT segment_name, tablespace_name, owner, status
2 FROM dba_rollback_segs;
```

SEGMENT_NAME	TABLESPACE_NAME	OWNER	STATUS
SYSTEM	SYSTEM	SYS	ONLINE
RBS1	RBS	SYS	ONLINE
RBS2	RBS	SYS	ONLINE
RBS3	RBS	SYS	OFFLINE

4 rows selected.

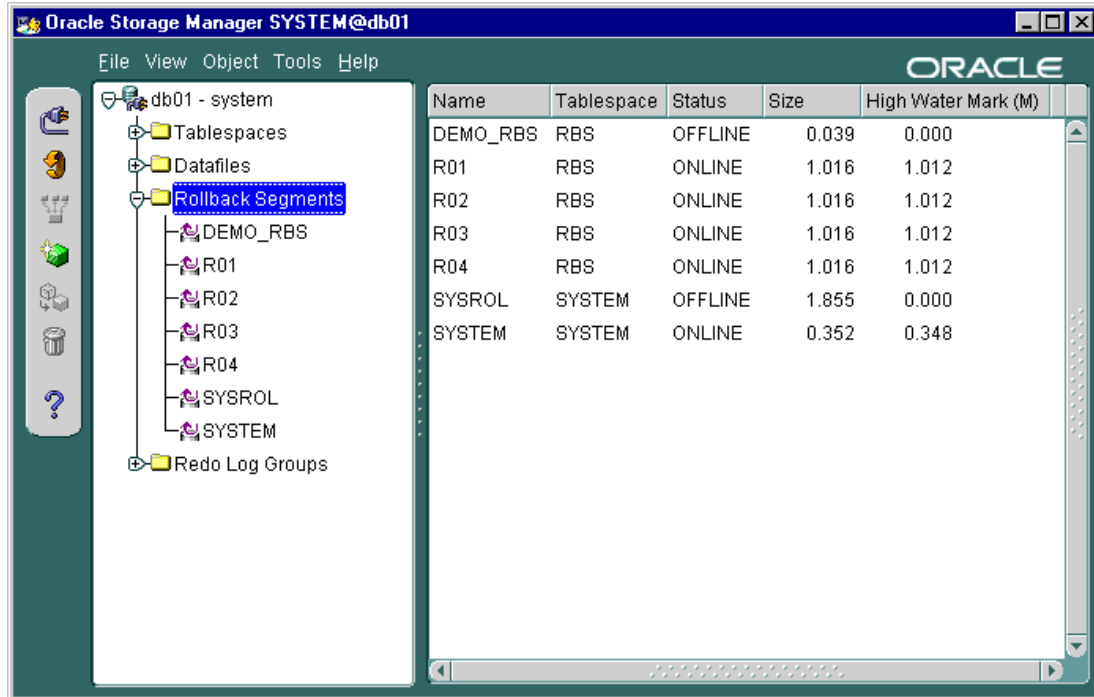
Information about rollback segments that are offline can only be seen in DBA_ROLLBACK_SEGS. The dynamic performance views only show rollback segments that are online.

The OWNER column specifies the type of a rollback segment:

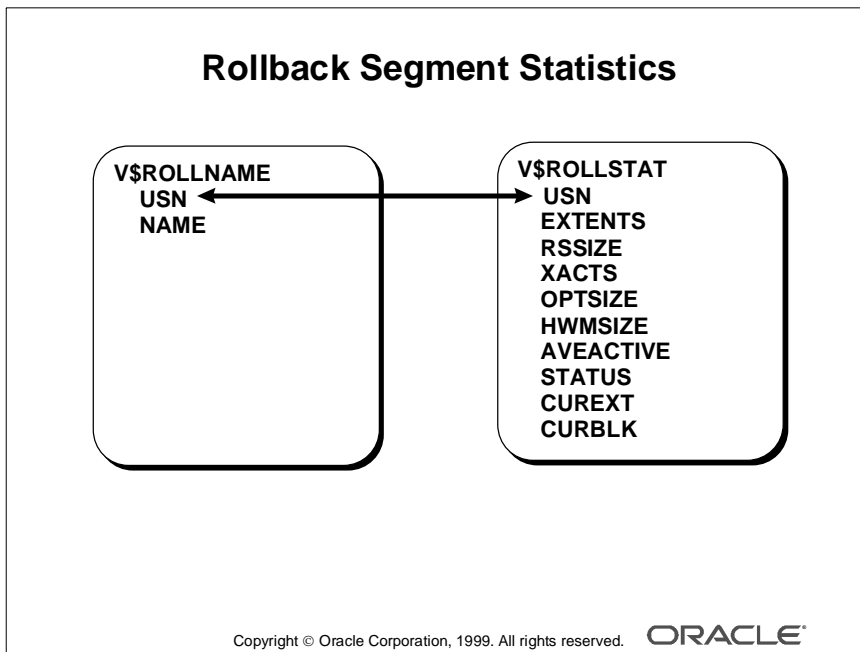
- SYS refers to a private rollback segment.
- PUBLIC refers to a public rollback segment.

How to Use Oracle Enterprise Manager to Drop a Rollback Segment

- 1 Launch Storage Manager and connect directly to the database:
Start—>Programs—>Oracle - *EMV2 Home*—>DBA Management Pack
—>Storage Manager
- 2 Enter the login information, and click OK.
- 3 Select the Rollback Segments node to view a summary of all the rollback segments in the database.



- 4 Expand the Rollback Segments folder.
- 5 Select the rollback segment to view extent information and the type of the rollback segment.



V\$ROLLSTAT and V\$ROLLNAME

Join the V\$ROLLSTAT and V\$ROLLNAME views to obtain the statistics of the rollback segments currently used by the instance.

Example

```

SQL> SELECT n.name, s.extents, s.rssize, s.optsize,
2      s.hwmsize, s.xacts, s.status
3      FROM v$rollname n, v$rollstat s
4      WHERE n.usn = s.usn;
  
```

NAME	EXTENTS	RSSIZE	OPTSIZE	HWMSIZE	XACTS	STATUS
SYSTEM	43	2199552		2199552	0	ONLINE
RBS1	20	202752	204800	417792	0	ONLINE
RBS2	4	38912		38912	0	PENDING
						OFFLINE

3 rows selected.

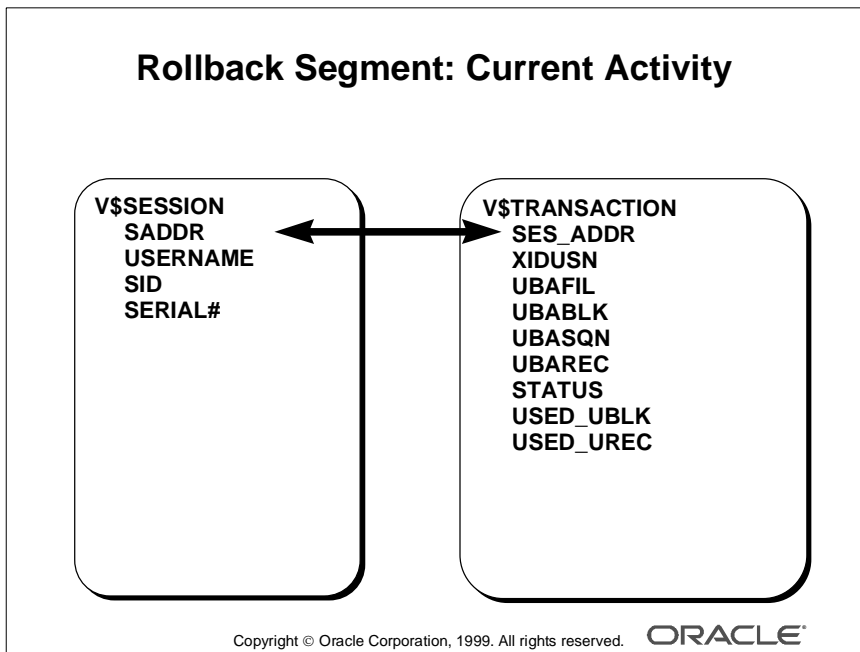
Note: The value of OPTIMAL can be obtained only from V\$ROLLSTAT view.

V\$ROLLSTAT and V\$ROLLNAME (continued)

The columns in V\$ROLLSTAT include:

Column	Description
USN	Rollback (undo) segment number; join with V\$ROLLNAME.USN to get the name
EXTENTS	Number of extents in the rollback segment
RSSIZE	Current size of the segment in bytes
XACTS	Number of transactions using this segment
OPTSIZE	OPTIMAL value for the rollback segment
HWMSIZE	High-water mark; the maximum size in bytes to which the segment grew since startup
AVEACTIVE	Current size of active extents, averaged over time
STATUS	Status of the rollback segment: <ul style="list-style-type: none"> • ONLINE indicates that the rollback segment is available for use. • PENDING OFFLINE indicates that the command to take a rollback segment offline has been executed and there are some active transactions using the segment. The rollback segment will be taken offline as soon as all transactions using the rollback segment have been completed. • OFFLINE indicates that the rollback segment cannot be used.
CUREXT	Current location of the rollback segment head; the extent and block number
CURBLK	

Note: If a rollback segment is **PENDING OFFLINE**, the status shows up as **ONLINE** in DBA_ROLLBACK_SEGS.



V\$SESSION and V\$TRANSACTION

To check the use of rollback segment by currently active transactions, join the V\$TRANSACTION and V\$SESSION views.

Example

```
SQL> SELECT s.username, t.xidusn, t.ubafil,
2 t.ubablk, t.used_ublk
3 FROM v$session s, v$transaction t
4 WHERE s.saddr = t.ses_addr;
```

USERNAME	XIDUSN	UBAFIL	UBABLK	USED_UBLK
SYSTEM	2	2	7	1
SCOTT	1	2	163	1

2 rows selected.

V\$TRANSACTION and V\$SESSION (continued)

The relevant columns in V\$TRANSACTION and their descriptions are shown below:

Column	Description
SES_ADDR	Address of the session; can be joined to V\$SESSION.SADDR
XIDUSN	Rollback (undo) segment number used by the transaction; used as part of the transaction ID
UBAFIL	Specify the current location in the rollback segment that the transaction is writing to
UBABLK	
UBASQN	
UBAREC	
USED_UBLK	Number of blocks of undo generated by the transaction
START_UEXT	Rollback segment extent from which the transaction started writing
START_UBAFIL	Rollback segment file number from which the transaction started writing
START_UBABLK	Rollback segment block number from which the transaction started writing

Planning Rollback Segments

Planning Rollback Segments: Number

- **OLTP**
 - Many small rollback segments
 - Four transactions per rollback segment
 - Up to ten transactions per rollback segment
- **Batch**
 - Few large rollback segments
 - One per transaction

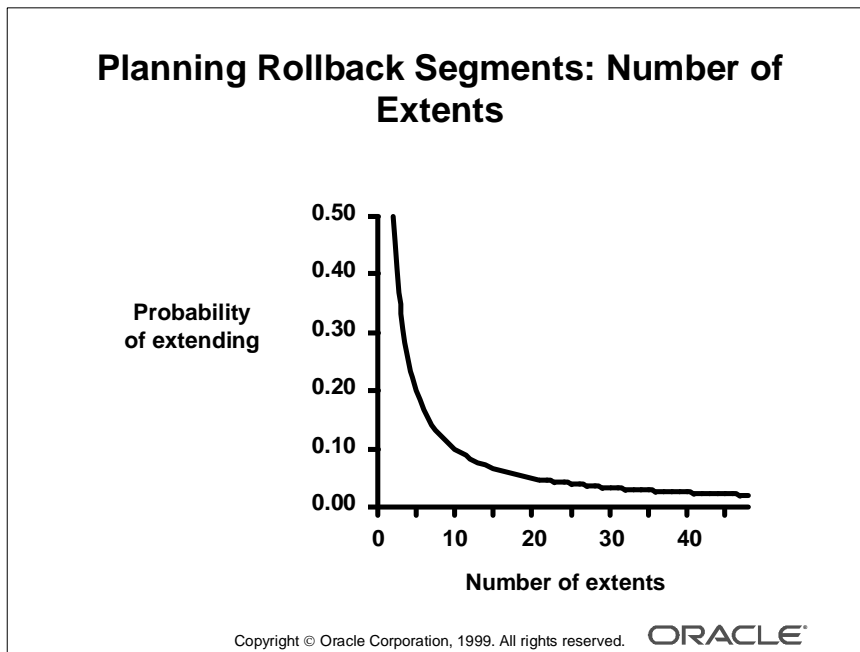
Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Number of Rollback Segments

The header block of a rollback segment contains transaction table entries that define the state of each transaction. Every transaction that uses a rollback segment needs to update the transaction table frequently. This could cause contention on the header, especially in an online transaction processing (OLTP) environment. Because OLTP environments typically use short transactions, many small rollback segments are recommended in this environment. If possible, create one rollback segment for every four concurrent transactions.

Batch environments generally run fewer jobs that may need to carry out several changes. These jobs require large rollback segments. Hence, in a batch environment, allow for the growth of the rollback segments by creating them in large tablespaces.



Size of a Rollback Segment

The number of bytes required to store information that is needed in case of rollback depends on two things:

- The type of transaction being performed (insert, update, delete, and so on)
- The actual data being processed

In general, inserting a given record into a table generates less undo than deleting the same record. Inserts need to store only the ROWID in the rollback, while deletes need to store the actual row itself.

You can estimate the size of the rollback segment by running the longest transaction expected and checking the size of the rollback segment or the amount of rollback (undo) generated. The courses *Enterprise DBA Part 2: Performance Tuning* and *Oracle8i: SQL Statement Tuning* discuss monitoring statistics.

Number of Extents

It has been found that a dynamic extension of a rollback segment can be minimized by creating rollback segments with a large number of extents. Creating rollback segments with MINEXTENTS=20 is recommended to reduce the possibility of extension.

Troubleshooting Rollback Segment Problems

Rollback Segment Problems

- Insufficient space for transactions
- Read-consistency error
- Blocking session
- Error in taking tablespace offline

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Insufficient Space for Transactions

- **No space in tablespace**
 - **Extend data files**
 - **Allow automatic extension of data files**
 - **Add data files**
- **MAXEXTENTS reached for segment**
 - **Increase MAXEXTENTS**
 - **Re-create segments with larger extent sizes**

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE

Possible Causes

A transaction uses a single rollback segment and may fail if there is insufficient space in the rollback segment (ORA-01562). This could be caused by one of the following:

- There is insufficient space in the tablespace for the rollback segments to extend (ORA-01560).
- The number of extents in the rollback segment has reached MAXEXTENTS and additional extents cannot be allocated (ORA-01628).

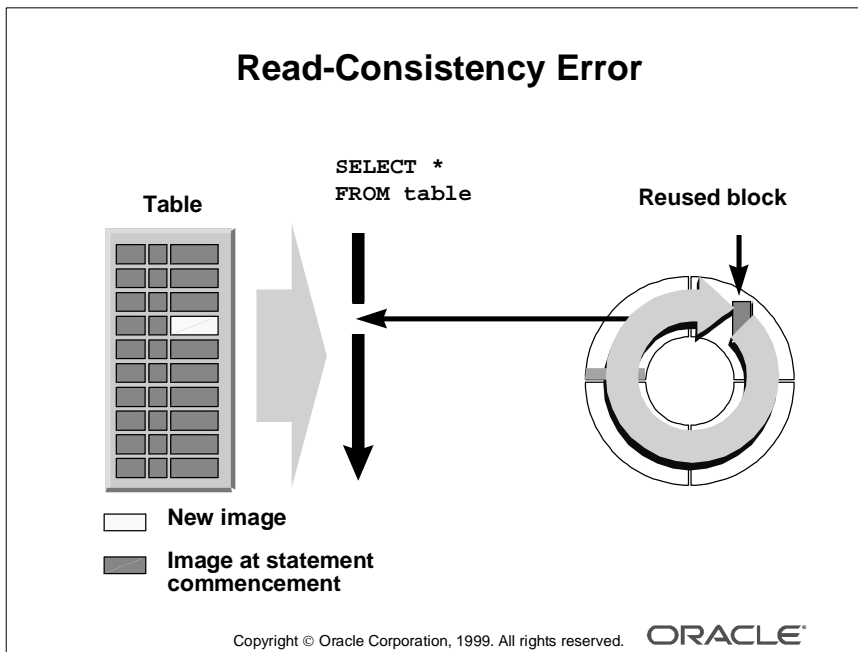
Solution

If the tablespace does not have free space, increase the space available by:

- Setting OPTIMAL to ensure that a single rollback segment does not use all of the free space in the tablespace
- Shrinking rollback segments back to their optimal size
- Increasing the size of the tablespace

If a rollback segment cannot allocate more extents because the limit imposed by MAXEXTENTS has been reached:

- Increase MAXEXTENTS for the rollback segment
- Drop and re-create the rollback segment with larger extent sizes to avoid a recurrence of the problem



Possible Causes

To provide read consistency, the Oracle server guarantees that changes made by other users that are not committed when the statement begins or are made after the statement begins execution will not be seen by the statement. If the Oracle server cannot construct a read-consistent image of data, the user will receive an ORA-01555 SNAPSHOT TOO OLD error. This error can occur when the transaction that made the change has already committed and:

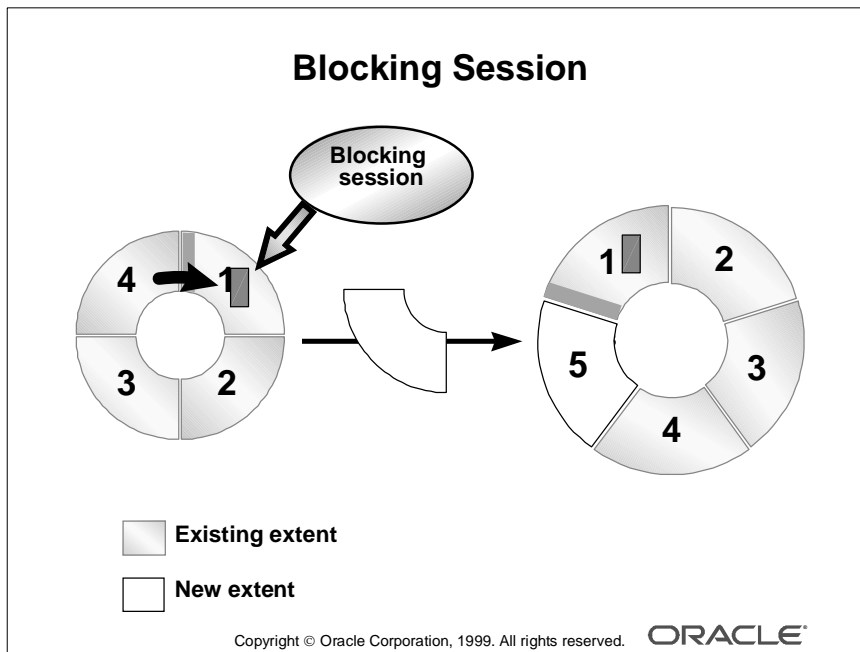
- The transaction slot in the rollback header has been reused
- The before-image in the rollback segment has been overwritten by another transaction

Solution

Read-consistency errors can be minimized by ensuring that rollback segments are created with:

- A higher MINEXTENTS value
- Larger extent sizes
- A higher OPTIMAL value

Note that these errors cannot be avoided by increasing MAXEXTENTS.



Possible Causes

When an extent in a rollback segment is full, the Oracle server attempts to reuse the next extent in the segment. If this new extent contains one active entry—that is, an entry by a transaction that is still active—it cannot be used. In these cases, a rollback segment allocates an additional extent. The transaction cannot skip an extent in the ring and continue writing to a subsequent extent. A transaction that has made only a few changes, but has been idle for a long time, could cause rollback segments to grow even though there are many free extents. In this situation, a lot of space is wasted and a database administrator may need to intervene to avoid excessive rollback segment growth.

Solution

Query the V\$ROLLSTAT, V\$SESSION, and V\$TRANSACTION views to find any blocking transactions.

Example

```
SQL> SELECT s.sid, s.serial#, t.start_time, t.xidusn, s.username
  2  FROM v$session s, v$transaction t, v$rollstat r
  3  WHERE s.saddr = t.ses_addr
  4  AND t.xidusn = r.usn
  5  AND ((r.curext = t.start_uext-1) OR
  6  ((r.curext = r.extents-1) AND t.start_uext=0));
  SID      SERIAL#  START_TIME          XIDUSN  USERNAME
  ---      -
    9         27  10/30/97 21:10:41         2    SYSTEM
1 row selected.
```

Check to see if the transaction can be ended by the user. If not, it may be necessary to kill the session.

Error in Taking a Tablespace Offline

Cannot take tablespace containing active RBS offline

- 1. Determine which rollback segments are in the tablespace**
- 2. Take all of these rollback segments offline**
- 3. Find active transactions using these rollback segments**
- 4. Find the session ID and serial number**
- 5. Terminate the session if necessary**
- 6. Take the tablespace offline**

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

Problem Diagnosis and Resolution

If a tablespace contains one or more active rollback segments, it cannot be taken offline. The session executing the statement will receive an ORA-01546 error message.

Solution

Perform the following steps:

- 1** Query DBA_ROLLBACK_SEGS to find which rollback segments are in the tablespace.
- 2** Take all rollback segments in the tablespace offline.
- 3** Check V\$TRANSACTION to find which transactions are currently using these rollback segments.
- 4** Use V\$SESSION to obtain the username and session information.
- 5** Kill the session or have the user end the transaction.
- 6** Take the tablespace offline.

Summary

Summary

In this lesson, you should have learned how to:

- **List rollback segments usage:**
 - Rollbacks
 - Read consistency
 - Recovery
- **Create adequate rollback segments**
- **Troubleshoot rollback segment problems**

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Quick Reference

Context	Reference
Initialization parameters	ROLLBACK_SEGMENTS TRANSACTIONS TRANSACTIONS_PER_ROLLBACK_SEGMENTS MAX_ROLLBACK_SEGMENTS
Dynamic performance views	V\$ROLLNAME V\$ROLLSTAT V\$TRANSACTION V\$SESSION
Data dictionary views	DBA_ROLLBACK_SEGS
Commands	CREATE ROLLBACK SEGMENT ALTER ROLLBACK SEGMENT ... ONLINE ALTER ROLLBACK SEGMENT ... STORAGE ALTER ROLLBACK SEGMENT ... SHRINK ALTER ROLLBACK SEGMENT ... OFFLINE DROP ROLLBACK SEGMENT SET TRANSACTION USE ROLLBACK SEGMENT

Managing Tables

Objectives

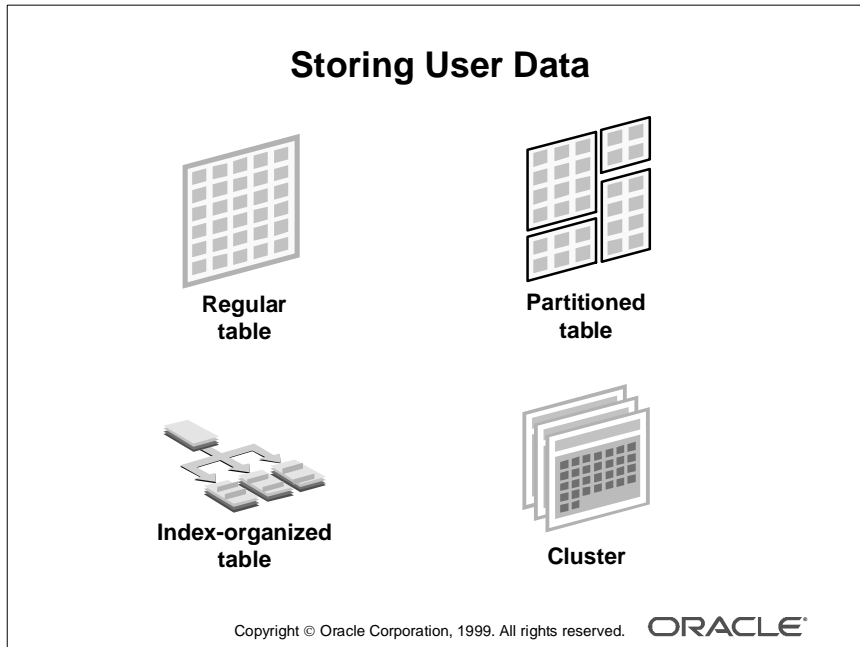
Objectives

After completing this lesson, you should be able to do the following:

- **Create tables using appropriate storage settings**
- **Control the space used by tables**
- **Analyze tables to check integrity and migration**
- **Retrieve information about tables from the data dictionary**
- **Convert between different formats of ROWID**

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

Overview



Using Different Methods for Storing User Data

There are several methods for storing user data in an Oracle database. The data can be stored in one of the following:

- Regular tables
- Partitioned tables
- Index-organized tables
- Clustered tables

Note: Partitioned tables, index-organized tables, and clustered tables are covered in other courses.

Regular Table

A regular table (generally referred to as a “table”) is the most commonly used form of storing user data. This is the default table and is the main focus of discussion in this lesson. A database administrator has very limited control over the distribution of rows in an unclustered table. Rows may be stored in any order depending on the activity on the table.

Partitioned Table

A partitioned table enables the building of scalable applications. It has the following characteristics:

- A partitioned table has one or more partitions, each of which stores rows that have been partitioned using range partitioning, hash partitioning, or composite partitioning.
- Each partition in a partitioned table is a segment and can be located in a different tablespace.
- Partitions are useful for large tables that can be queried or manipulated using several processes concurrently.
- Special commands are available to manage partitions within a table.

Index-Organized Table

An index-organized table is like a regular table with a primary key index on one or more of its columns. However, instead of maintaining two separate storage spaces for the table and a B-tree index, an index-organized table only maintains a single B-tree containing the primary key of the table and other column values.

Index-organized tables provide fast key-based access to table data for queries involving exact match and range searches.

Also, storage requirements are reduced because key columns are not duplicated in the table and index. The remaining non-key columns are stored in the index unless the index entry gets very large; in that case, the Oracle server provides an **OVERFLOW** clause to handle the problem.

Clustered Table

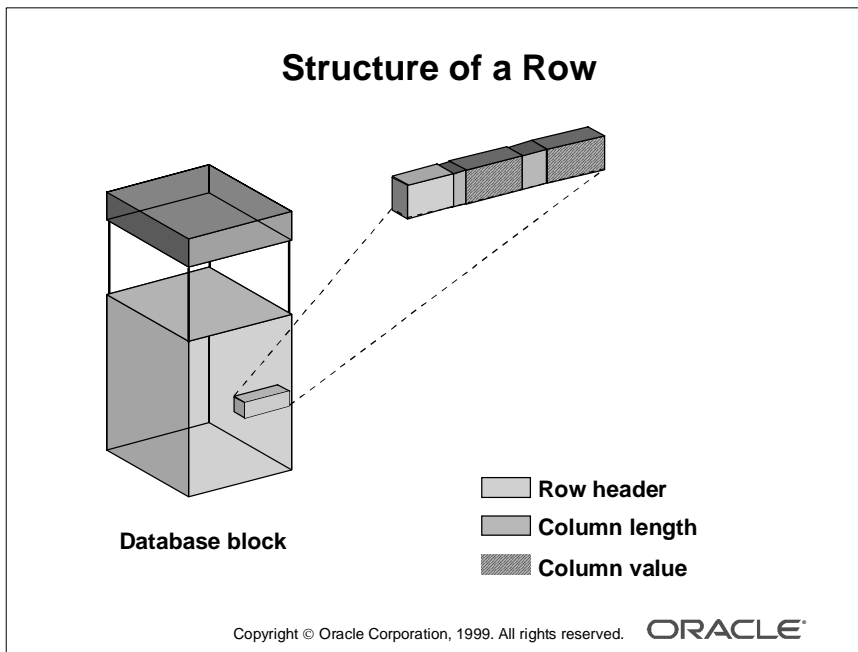
A clustered table provides an optional method for storing table data. A cluster is made up of a group of tables that share the same data blocks, which are grouped together because they share common columns and are often used together.

Clusters have the following characteristics:

- Clusters have a *cluster key*, which is used to identify the rows that need to be stored together.
- The cluster key can consist of one or more columns.
- Tables in a cluster have columns that correspond to the cluster key.
- Clustering is a mechanism that is transparent to the applications using the tables. Data in a clustered table can be manipulated as though it were stored in a regular table.

Clustered Table (continued)

- Updating one of the columns in the cluster key may entail physically relocating the row.
- The cluster key is independent of the primary key. The tables in a cluster can have a primary key, which may be the cluster key or a different set of columns.
- Clusters are usually created to improve performance. Random access to clustered data may be faster, but full table scans on clustered tables are generally slower.



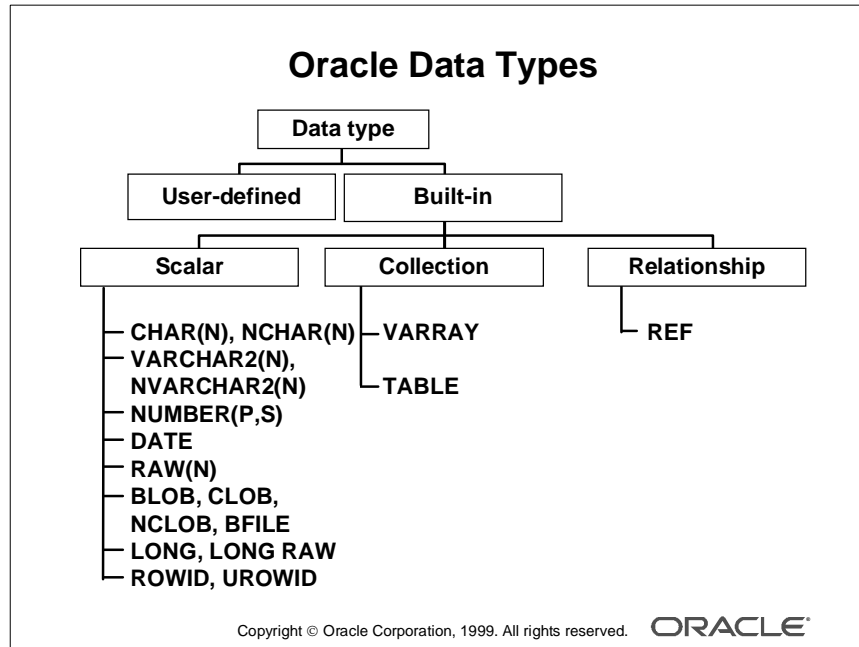
Row Format and Size

Row data is stored in database blocks as variable-length records. Columns for a row are generally stored in the order in which they are defined, and any trailing NULL columns are not stored. Each row in a table can have a different number of columns. Each row in a table has:

- A row header: Used to store the number of columns in the row, the chaining information, and the row lock status
- Row data: For each column, the Oracle server stores the column length and value (One byte is needed to store the column length if the column cannot exceed 250 bytes. A column that can be longer needs three length bytes. The column value is stored immediately following the column length bytes.)

Adjacent rows do not need any space between them. Each row in the block has a slot in the row directory. The directory slot points to the beginning of the row.

Oracle Data Types



Oracle Built-in Data Types

The Oracle server provides several built-in data types to store scalar data, collections, and relationships.

Scalar Data Types

Character Data Character data can be stored as either fixed-length or variable-length strings in the database.

Fixed-length character data types, such as CHAR and NCHAR, are stored with padded blanks. NCHAR is a national language-supported (NLS) data type that enables the storage of either fixed-width or variable-width character sets. The maximum size is determined by the number of bytes required to store one character, with an upper limit of 2,000 bytes per row. The default is 1 character or 1 byte, depending on the character set.

Variable-length character data types use only the number of bytes needed to store the actual column value, and can vary in size for each row, up to 4,000 bytes. VARCHAR2 and NVARCHAR2 are examples of variable-length character data types.

Scalar Data Types (continued)

Numeric Data Numbers in an Oracle database are always stored as variable-length data. They can store up to 38 significant digits. Numeric data types require:

- One byte for the exponent
- One byte for every two significant digits in the mantissa
- One byte for negative numbers if the number of significant digits is less than 38 bytes

DATE Data Type The Oracle server stores dates in fixed-length fields of seven bytes. An Oracle DATE always includes the time.

RAW Data Type This data type enables the storage of small binary data. The Oracle server does not perform character set conversion when RAW data is transmitted across machines in a network or if RAW data is moved from one database to another using Oracle utilities. The number of bytes needed to store the actual column value vary in size for each row, up to 2,000 bytes.

Data Types for Storing Large Objects

LONG, LONG RAW	LOB
Single column per table	Multiple columns per table
Up to 2 gigabytes	Up to 4 gigabytes
SELECT returns data	SELECT returns locator
Data stored in-line	Data stored in-line or out-of-line
No object type support	Supports object types
Sequential access to chunks	Random access to chunks

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

Scalar Data Types for Storing Large Objects (LOBs)

Oracle provides six data types for storing LOBs:

- CLOB and LONG for large fixed-width character data
- NCLOB for large fixed-width national character set data
- BLOB and LONG RAW for storing unstructured data
- BFILE for storing unstructured data in operating system files

LONG and LONG RAW data types were previously used for unstructured data, such as binary images, documents, or geographical information, and are primarily provided for backward compatibility. These data types are superseded by the LOB data types. LOB data types are distinct from LONG and LONG RAW, and they are not interchangeable. LOBs will not support the LONG application programming interface (API), and vice versa.

Comparing LONG and LOB Data Types

It is beneficial to discuss LOB functionality in comparison to the older types. Below, LONGs refers to LONG and LONG RAW, and LOBs refer to all LOB data types.

LOBs enable multiple LOB columns per table or attributes in an object type; LONGs enable only one.

Comparing LONG and LOB Data Types (continued)

The maximum size of LONGs is 2 gigabytes; LOBs can be up to 4 gigabytes.

Upon retrieval, LOBs return the locator; LONGs return the data.

LOBs store a locator in the table and the data elsewhere, unless the size is less than the maximum size for a VARCHAR2 data type, which is 4,000 bytes; LONGs store all data in-line. In addition, LOBs allow data to be stored in a separate segment and tablespace, or in a host file.

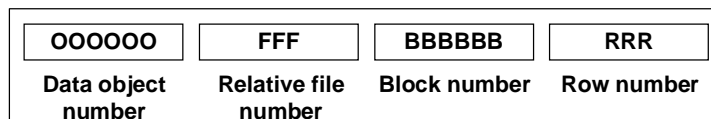
LOBs support object type attributes (except NCLOBs); LONGs do not.

LONGs are primarily stored as chained row pieces, with a row piece in one block pointing to the next row piece stored in another block. Therefore, they need to be accessed sequentially. In contrast, LOBs support random piece-wise access to the data through a file-like interface.

ROWID Data Type

- Unique identifier for a row
- Used to locate a row

ROWID Format



Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

ROWID and UROWID Data Type

ROWID is a pseudo-column that can be queried along with other columns in a table. It has the following characteristics:

- ROWID is a unique identifier for each row in the database.
- ROWID is not stored explicitly as a column value.
- Although the ROWID does not directly give the physical address of a row, it can be used to locate the row.
- ROWID provides the fastest means of accessing a row in a table.
- ROWIDs are stored in indexes to specify rows with a given set of key values.

With release 8.1, the Oracle server provides a new single data type called the universal rowid, or UROWID. It supports rowids of foreign tables (tables other than Oracle's) and can store all kinds of rowids. The value of the parameter COMPATIBLE must be set to 8.1 or higher to use UROWID.

ROWID Format

ROWID needs 10 bytes of storage on disk and is displayed using 18 characters. It consists of the following components:

- *Data object number* is assigned to each data object, such as a table or index, when it is created, and it is unique within the database.
- *Relative file number* is unique to each file within a tablespace.

ROWID Format (continued)

- *Block number* represents the position of the block containing the row within the file.
- *Row number* identifies the position of the row directory slot in the block header.

Internally, the data object number needs 32 bits, the relative file number needs 10 bits, the block number needs 22 bits, and the row number needs 16 bits, adding up to a total of 80 bits or 10 bytes.

ROWID is displayed using a base-64 encoding scheme, which uses six positions for the data object number, three positions for the relative file number, six positions for the block number, and three positions for the row number. The base-64 encoding scheme uses the characters “A-Z,” “a-z,” “0-9,” “+,” and “/”—a total of 64 characters, as in the example below:

```
SQL> SELECT id, ROWID FROM summit.department;
      ID ROWID
-----
10 AAADC4AACAAAAMAAAA
31 AAADC4AACAAAAMAAAB
32 AAADC4AACAAAAMAAAC
33 AAADC4AACAAAAMAAAD
34 AAADC4AACAAAAMAAAE
35 AAADC4AACAAAAMAAAF
41 AAADC4AACAAAAMAAAG
42 AAADC4AACAAAAMAAAH
43 AAADC4AACAAAAMAAAI
44 AAADC4AACAAAAMAAAJ
45 AAADC4AACAAAAMAAAK
50 AAADC4AACAAAAMAAAL
```

In this example:

- AAADC4 is the data object number.
- AAC is the relative file number.
- AAAAMA is the block number.
- AAA is the row number for the department with ID=10.

Locating a Row Using ROWID

Because a segment can only reside in one tablespace, using the data object number, the Oracle server can determine the tablespace that contains a row.

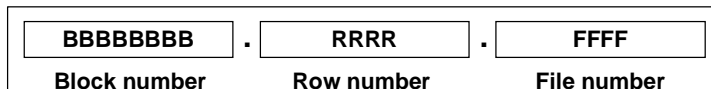
The relative file number within the tablespace is used to locate the file, the block number is used to locate the block containing the row, and the row number is used to locate the row directory entry for the row.

The row directory entry can be used to locate the beginning of the row.

Thus, ROWID can be used to locate any row within a database.

Restricted ROWID

- Can identify rows within a segment
- Needs less space



Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE®

Using Restricted ROWID in Oracle7 and Earlier

Versions of Oracle prior to the Oracle8 server used the restricted ROWID format. A restricted ROWID used only six bytes internally and did not contain the data object number. This format was acceptable in Oracle7 or an earlier release because the file numbers were unique within a database. Thus, earlier releases did not permit more than 1,022 data files.

Even though Oracle8 removed this restriction by using tablespace-relative file numbers, the restricted ROWID is still used in objects like nonpartitioned indexes on nonpartitioned tables where all the index entries refer to rows within the same segment.

Collections

- **Collections are objects that contain objects.**
- **VARRAYs are ordered sets of elements containing a count and a limit.**
- **Nested tables are tables with a column or variable of the TABLE data type.**



VARRAY



**Nested
table**

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE

Collection Data Types

Two types of collection data types are available to store data that is repetitive for a given row in a table. Prior to Oracle8i, the Objects option was needed to define and use collections. A brief discussion of these types follows.

Varying Arrays (VARRAYs) Varying arrays are useful to store lists that contain a small number of elements, such as phone numbers for a customer.

VARRAYs have the following characteristics:

- An array is an ordered set of data elements.
- All elements of a given array are of the same data type.
- Each element has an index, which is a number corresponding to the position of the element in the array.
- The number of elements in an array is the size of the array.
- The Oracle server allows arrays to be of variable size, which is why they are called VARRAYs, but the maximum size must be specified when declaring the array type.

Collection Data Types (continued)

Nested Tables Nested tables provide a means of defining a table as a column within a table. They can be used to store sets that may have a large number of records, such as number of items in an order.

Nested tables generally have the following characteristics:

- A nested table is an unordered set of records or rows.
- The rows in a nested table have the same structure.
- Rows in a nested table are stored separately from the parent table, with a pointer from the corresponding row in the parent table.
- Storage characteristics for the nested table can be defined by the database administrator.
- There is no predetermined maximum size for a nested table.

Relationship Data Types (REFs)

Relationship types are used as pointers within the database. The use of these types requires the Objects option. As an example, each item that is ordered could point to or reference a row in the PRODUCTS table, without having to store the product code.

User-Defined Data Types

The Oracle server allows a user to define abstract data types and use them within the application. The use of this feature requires the Objects option.

Creating a Table

Creating a Table

```
CREATE TABLE employee(
  id          NUMBER(7),
  last_name   VARCHAR2(25),
  dept_id     NUMBER(7))
PCTFREE 20 PCTUSED 50
STORAGE(INITIAL 200K NEXT 200K
PCTINCREASE 0   MAXEXTENTS 50)
TABLESPACE data;
```

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE™

Syntax

Use the following command to create a table:

```
CREATE TABLE [schema.] table
(column datatype [ , column datatype ] ...)
[TABLESPACE tablespace ]
[ PCTFREE integer ]
[ PCTUSED integer ]
[ INITTRANS integer ]
[ MAXTRANS integer ]
[ STORAGE storage-clause ]
[LOGGING | NOLOGGING]
[CACHE | NOCACHE] ]
```

where:	schema	is the owner of the table
	table	is the name of the table
	column	is the name of the column
	data type	is the data type of the column

Syntax (continued)

TABLESPACE	identifies the tablespace where the table will be created
PCTFREE	is the amount of space reserved in each block (in a percentage of total space minus the block header) for rows to grow in length
PCTUSED	determines lower limit of space used on a block (after it fills to PCTFREE) before it becomes available for further row inserts
INITRANS	specifies the number of transaction entries preallocated in each block (The default is 1.)
MAXTRANS	limits the number of transaction entries that can be allocated to each block (The default is 255.)
STORAGE	identifies the storage clause that determines how extents will be allocated to the table
LOGGING	specifies that the creation of the table will be logged in the redo log file (It also specifies that all subsequent operations against the table are logged. This is the default.)
NOLOGGING	specifies that the creation of the table and certain types of data loads will not be logged in the redo log file
CACHE	specifies that the blocks retrieved for this table are placed at the most recently used end of the LRU list in the buffer cache even when a full table scan is performed
NOCACHE	specifies that the blocks retrieved for this table are placed at the least recently used end of the LRU list in the buffer cache when a full table scan is performed

Syntax (continued)

Note

- Generally, tables should be created with a primary key. The maintenance of constraints is discussed in the lesson “Maintaining Data Integrity.”
- If MINIMUM EXTENT has been defined for the tablespace, the extent sizes for the table will be rounded up to the next higher multiple of the MINIMUM EXTENT value.
- If the [NO]LOGGING clause is omitted, the logging attribute of the table defaults to the logging attribute of the tablespace in which it resides.
- If MINEXTENTS is specified to a value greater than one and the tablespace contains more than one data file, the extents will be spread across the different files in the tablespace.

Copying an Existing Table

Use the CREATE TABLE command with a subquery to copy an existing table in full or in part.

The simplified syntax for this command is:

```
CREATE TABLE [schema.]table
[ LOGGING | NOLOGGING ]
...
AS
subquery
```

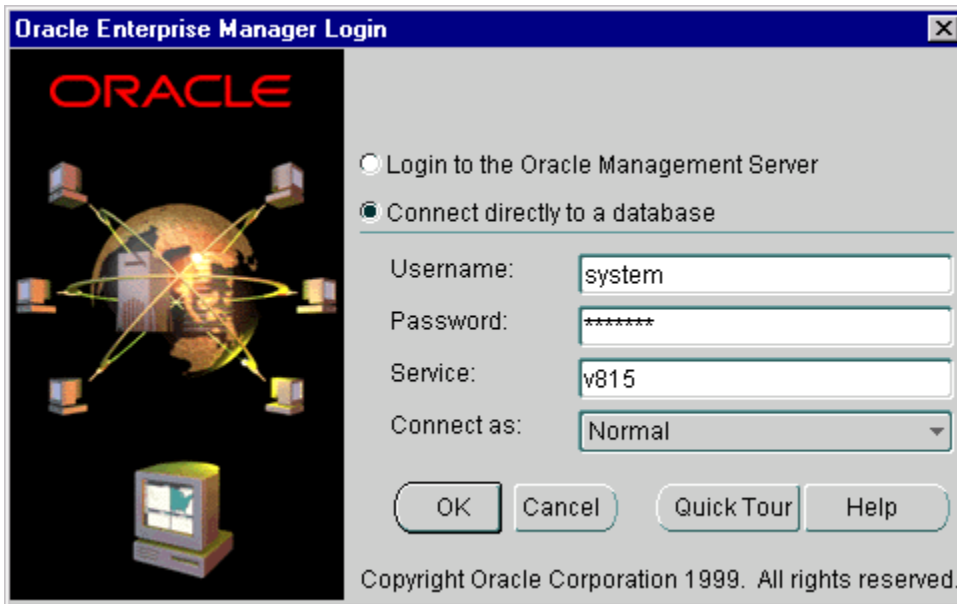
The other clauses such as TABLESPACE, STORAGE, and block utilization parameters can be specified while creating a table based on another table. Use the NOLOGGING clause to suppress generation of redo log entries and speed up the creation of the table.

Constraints, triggers, and table privileges are not copied to the new table that is created in this manner. If a column was defined as NOT NULL in the original table, the corresponding column in the new table will also be defined as NOT NULL.

How to Create a Table Using Schema Manager

Try It 11-1

- 1 Launch Schema Manager:
Start—>Programs—>Oracle - *EMV2 Home*—>DBA Management Pack
—>Schema Manager
- 2 Select the option to connect directly to a database. Enter the administrator system, the password manager, the service name your working database, and click OK.



- 3 Select Object—>Create from the menu bar.
- 4 Choose Table in the list of objects, select the Use Wizard option, and click Create.
- 5 Enter your table information in the Table Wizard, such as table name, tablespace, owner, columns, and data types and sizes. Click Finish.
- 6 Expand the Tables folder to verify that your table was created.

Alternatively, select an existing table from the navigator, and use Object—>Create Like to create a new table with the same column and storage characteristics as an existing table.

Other Options While using Oracle Schema Manager, you also have the option to let the tool automatically define the storage and block utilization parameters based on an estimate of the initial volume, the growth rate, and the DML activity on the table.

Temporary Tables

- The rows are private to the session.

```
CREATE GLOBAL TEMPORARY TABLE employee_temp  
AS SELECT * FROM employee;
```

- Tables retain data only for the duration of a transaction or session.

```
ON COMMIT PRESERVE ROWS;
```

- DML locks are not acquired on the data.
- DMLs do not generate redo logs.

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE®

Temporary Tables

In addition to permanent tables, you can create temporary tables to hold session-private data that exists only for the duration of a transaction or session.

The `CREATE GLOBAL TEMPORARY TABLE` command creates a temporary table that can be transaction-specific or session-specific. For transaction-specific temporary tables, data exists for the duration of the transaction, while for session-specific temporary tables, data exists for the duration of the session. Data in a session is private to the session. Each session can only see and modify its own data. DML locks are not acquired on the data of the temporary tables. The clauses that control the duration of the rows are:

- `ON COMMIT DELETE ROWS` to specify that rows are only visible within the transaction
- `ON COMMIT PRESERVE ROWS` to specify that rows are visible for the entire session

You can create indexes, views, and triggers on temporary tables and you can also use the Export and Import utilities to export and import the definition of a temporary table. However, no data is exported, even if you use the `ROWS` option. The definition of a temporary table is visible to all sessions.

Creating a Table: Guidelines

- **Use a few standard extent sizes for tables to reduce tablespace fragmentation.**
- **Use locally managed tablespaces to avoid fragmentation.**
- **Use the CACHE clause for frequently used, small tables.**

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Guidelines for Creating a Table

- Place tables in a separate tablespace—not in the tablespace that has rollback segments, temporary segments, and indexes.
- Place tables in locally managed tablespaces to avoid fragmentation.
- Use a few standard extent sizes that are multiples of $5 \times \text{DB_BLOCK_SIZE}$ to minimize fragmentation.
- To improve performance of full table scans, align extent sizes with `DB_FILE_MULTIBLOCK_READ_COUNT`, which is an initialization parameter that defines how many blocks are requested by the server processes in each read call to the operating system while reading the whole table.
- Use the CACHE clause for small reference tables that are likely to be accessed very frequently.

Setting PCTFREE and PCTUSED

- **Compute PCTFREE**

$$\frac{(\text{Average Row Size} - \text{Initial Row Size}) * 100}{\text{Average Row Size}}$$

- **Compute PCTUSED**

$$100 - \text{PCTFREE} - \frac{\text{Average Row Size} * 100}{\text{Available Data Space}}$$

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Setting PCTFREE

A higher PCTFREE affords more room for updates within a database block. Set a higher value if the table contains:

- Columns that are initially NULL and later updated with a value
- Columns that are likely to increase in size as a result of an update

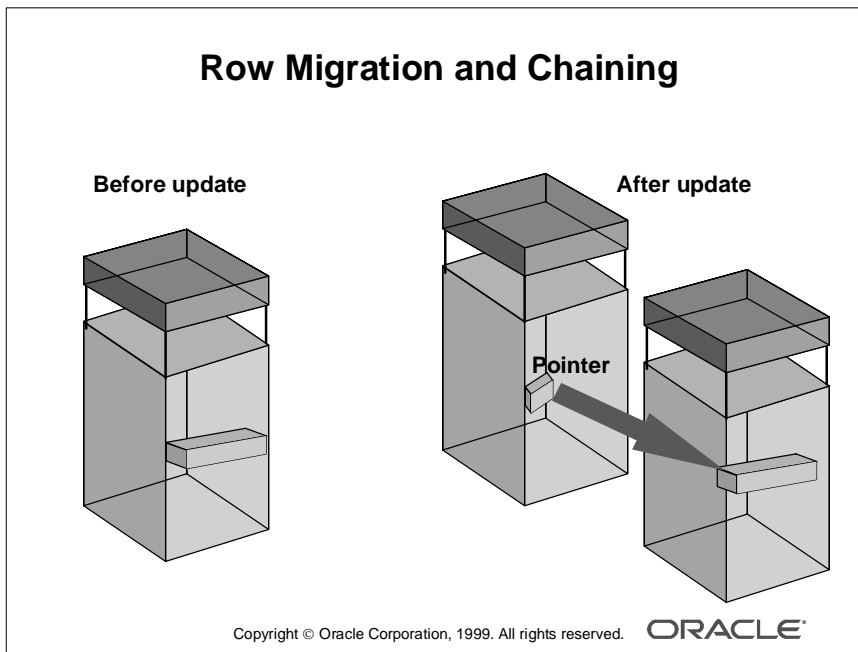
A higher PCTFREE will result in lower block density—each block can accommodate fewer rows.

The formula specified above ensures that there is enough free space in the block for row growth.

Setting PCTUSED

Set PCTUSED to ensure that the block is returned to the free list only when there is sufficient space to accommodate an average row. If a block on the free list does not contain sufficient space for inserting a row, the Oracle server looks up the next block on the free list. This linear scan continues until either a block with sufficient space is found or the end of the list is reached. Using the formula given reduces the time taken to scan the free list by increasing the probability of finding a block with the required free space.

Note: The value for average row size can be estimated using the ANALYZE TABLE command, which is discussed in a subsequent section.



Row Migration

If PCTFREE is set to a low value, there may be insufficient space in a block to accommodate a row that grows as a result of an update. When this happens, the Oracle server will move the entire row to a new block and leave a pointer from the original block to the new location. This process is referred to as row migration. When a row is migrated, I/O performance associated with this row decreases because the Oracle server must scan two data blocks to retrieve the row.

Row Chaining

Row chaining occurs when a row is too large to fit into any block. This might occur when the row contains columns that are very long. In this case, the Oracle server divides the row into smaller chunks called row pieces. Each row piece is stored in a block along with the necessary pointers to retrieve and assemble the entire row. Row chaining can be minimized by choosing a higher block size or by splitting the table into multiple tables with fewer columns, if possible.

Note: Row migration and row chaining are covered in more detail in the course *Enterprise DBA Part 2: Performance and Tuning*.

Controlling Space Used by Tables

Changing Storage and Block Utilization Parameters

```
ALTER TABLE summit.employee  
PCTFREE 30  
PCTUSED 50  
STORAGE(NEXT 500K  
MINEXTENTS 2  
MAXEXTENTS 100);
```

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE™

Changing Storage and Block Utilization Parameters

Some of the storage parameters and any of the block utilization parameters can be modified by using the ALTER TABLE command.

Syntax

```
ALTER TABLE [schema.]table  
{[ storage-clause ]  
[ PCTFREE integer ]  
[ PCTUSED integer ]  
[ INITTRANS integer ]  
[ MAXTRANS integer]}
```

Effects of Changing Storage Parameters

The parameters that can be modified and the implications of the modifications are as follows:

- **NEXT:** When the Oracle server allocates another extent for the table, the new value will be used. Subsequent extent sizes will increase by PCTINCREASE.

Effects of Changing Storage Parameters (continued)

- **PCTINCREASE:** A change in PCTINCREASE will be registered in the data dictionary. It will be used to recalculate NEXT when the next extent is allocated by the Oracle server. Consider a case where a table with two extents has NEXT=10K and PCTINCREASE=0. If PCTINCREASE is changed to 100, the third extent to be allocated will be 10K, the fourth extent will be 20K, and so on.
- **MINEXTENTS:** The value of MINEXTENTS can be changed to any value that is less than or equal to the current number of extents in the table. It will have no immediate effect on the table, but will be used if the table is truncated.
- **MAXEXTENTS:** The value of MAXEXTENTS can be set to any value equal to or greater than the current number of extents for the table.

Restrictions The value of INITIAL cannot be modified for a table.

The value of NEXT specified will be rounded to a value that is a multiple of the block size greater than or equal to the value specified.

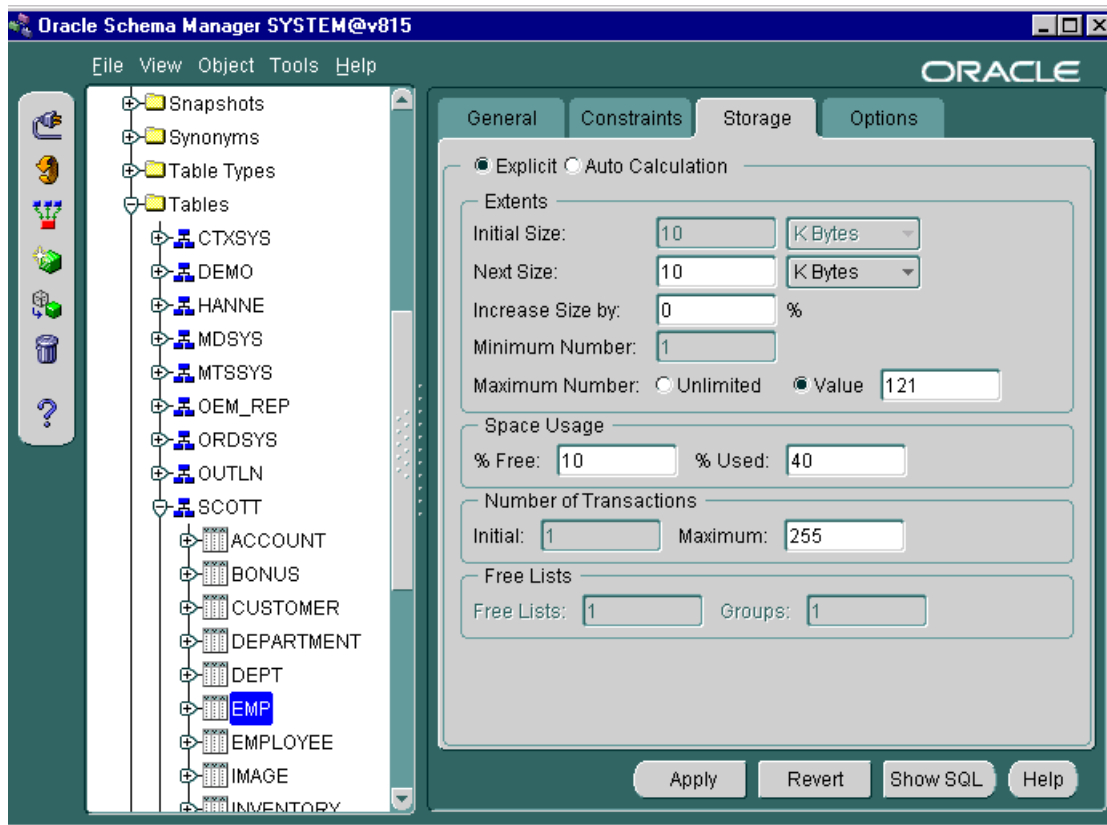
Block Utilization Parameters

Block utilization parameters may be changed to:

- Improve space utilization
- Minimize the possibility of migration

The effects of changing the block utilization parameters are as follows:

- **PCTFREE:** A change to PCTFREE will affect future inserts. Blocks that are not used for inserts because they had already been filled to $(100/\text{PCTFREE})$ will not be affected until they are back on the free list. They can only be placed on the free list if their use drops below PCTUSED.
- **PCTUSED:** Any change to PCTUSED will affect all the blocks in the table. If a row is updated or deleted, the block containing the row will be checked for its use and reused for inserts if the use is below PCTUSED.
- **INITTRANS:** A change to INITTRANS affects only new blocks.
- **MAXTRANS:** A change to MAXTRANS will affect all blocks in the table.



How to Use Oracle Enterprise Manager to Change Storage Parameters

- 1 Launch Schema Manager and connect directly to the database:
Start—>Programs—>Oracle - *EMV2 Home*—>DBA Management Pack
—>Schema Manager
- 2 Enter the login information, and click OK.
- 3 Expand the Tables folder.
- 4 Expand the username (or schema).
- 5 Select the table.
- 6 Modify the values in the Storage tab of the property sheet. Note that the minimum extents and initial number of transactions cannot be modified using this method.
- 7 Click Apply.

Manually Allocating Extents

```
ALTER TABLE summit.employee  
  ALLOCATE EXTENT(SIZE 500K  
  DATAFILE '/DISK3/DATA01.DBF');
```

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Manually Allocating Extents

Extents may need to be allocated manually:

- To control the distribution of extents of a table across files
- Before loading data in bulk to avoid dynamic extension of tables

Syntax

Use the following command to allocate an extent to a table:

```
ALTER TABLE [schema.]table  
  ALLOCATE EXTENT [ ([SIZE integer [K|M]]  
                    [ DATAFILE 'filename' ]) ]
```

If SIZE is omitted, the Oracle server will use the NEXT_EXTENT size from DBA_TABLES to allocate the extent.

The file specified in the DATAFILE clause must belong to the tablespace that the table belongs to. Otherwise, the statement will generate an error. If the DATAFILE clause is not used, the Oracle server will allocate the extent in one of the files in the tablespace containing the table.

Note: The NEXT_EXTENT value in DBA_TABLES will not be affected by manual extent allocation. The Oracle server will not recalculate the size of the next extent when this command is executed.

Nonpartitioned Table Reorganization

```
ALTER TABLE employee  
MOVE TABLESPACE data1;
```

- Moves data into a new segment while preserving indexes, constraints, privileges, and so on, on the table
- Is being used to move a table to a different tablespace or to reorganize extents

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE®

Relocating or Reorganizing a Table

Oracle8i provides the means of moving a nonpartitioned table without having to run the Export or Import utility. This is useful when:

- Moving a table from one tablespace to another
- Reorganizing the table to eliminate row migration

After moving a table, you will have to rebuild the indexes to avoid the following error:

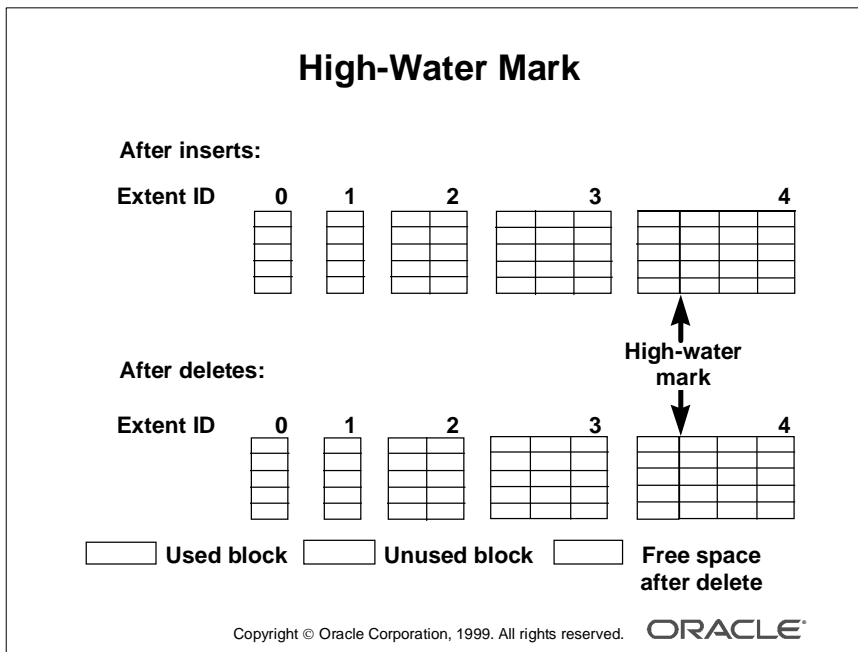
```
SQL> select * from employee where id=23;
```

```
select * from employee where id=23
```

```
*
```

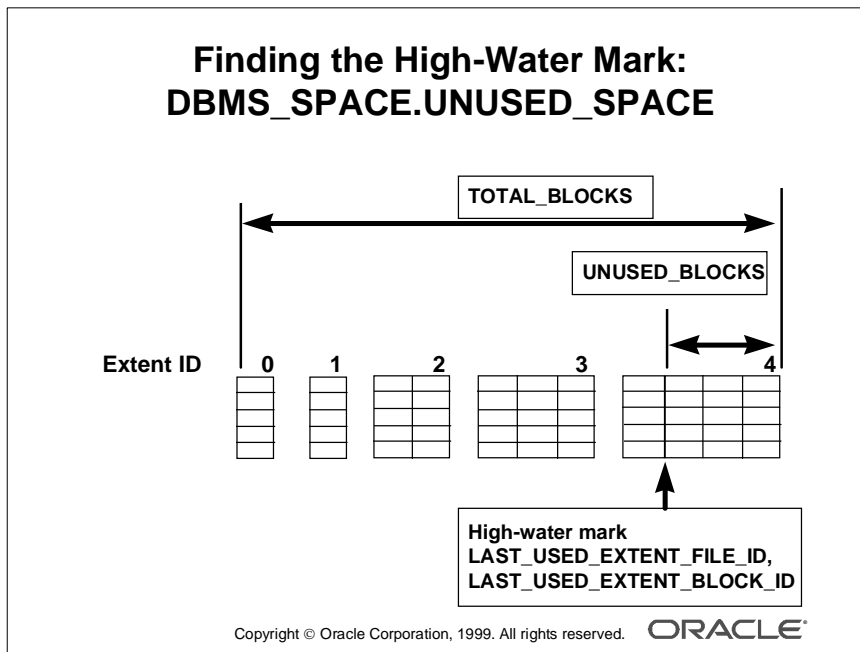
```
ERROR at line 1:
```

```
ORA-01502: index 'SUMMIT.EMPLOYEE_ID_PK' or partition of such  
index is in unusable state
```



What Is the High-Water Mark?

- The high-water mark for a table indicates the last block that was ever used for the table.
- As data is inserted into the table, the high-water mark is moved to mark the last block used.
- The high-water mark is not reset when rows are deleted from the table.
- The high-water mark is stored in the segment header of the table.
- When the Oracle server performs full table scans, it reads all the blocks up to the high-water mark.



How to Find the High-Water Mark

The DBMS_SPACE package contains a procedure that can be used to find the high-water mark and the number of blocks above the high-water mark. Although this information can be obtained from the data dictionary after analyzing the table, the DBMS_SPACE package enables faster access to the information without affecting the optimization behavior.

Syntax

The following PL/SQL block can be used to find and print the number of blocks allocated to a table and the number of unused blocks:

```
SQL> DECLARE
2   v_owner VARCHAR2(30) := 'SUMMIT' ;
3   v_segment_name VARCHAR2(30) := 'EMPLOYEE' ;
4   v_segment_type VARCHAR2(30) := 'TABLE' ;
5   v_total_blocks NUMBER;
6   v_total_bytes NUMBER;
7   v_unused_blocks NUMBER;
8   v_unused_bytes NUMBER;
9   v_last_used_extent_file_id NUMBER;
10  v_last_used_extent_block_id NUMBER;
```

Syntax (continued)

```
11 v_last_used_block NUMBER;
12 BEGIN
13 dbms_space.unused_space(v_owner,
14 v_segment_name,
15 v_segment_type,
16 v_total_blocks,
17 v_total_bytes,
18 v_unused_blocks,
19 v_unused_bytes,
20 v_last_used_extent_file_id,
21 v_last_used_extent_block_id,
22 v_last_used_block
23 );
24 dbms_output.put_line(INITCAP(v_segment_type)||' :
'|v_owner||'.'||v_segment_name);
25 dbms_output.put_line('Total Blocks :
'|TO_CHAR(v_total_blocks));
26 dbms_output.put_line('Blocks above HWM :
'|TO_CHAR(v_unused_blocks));
27 END;
28 /
```

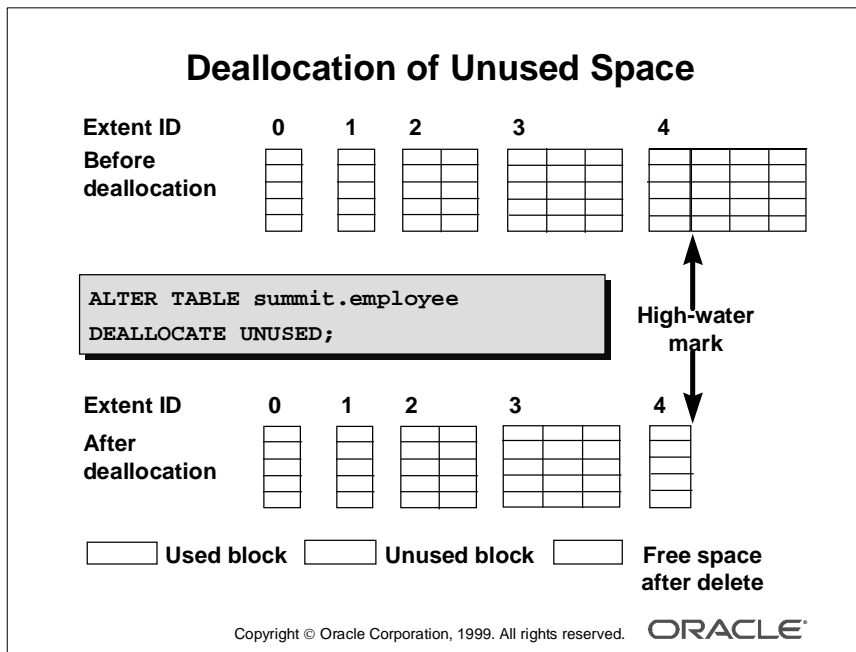
Statement processed.

Table : SUMMIT.EMPLOYEE

Total Blocks : 25

Blocks above HWM : 23

Note: The DBMS_SPACE package is created when the dbmsutil.sql and prvtutil.plb scripts are invoked by catproc.sql.



Deallocation of Unused Space

If large extents have been allocated to a table, but have not been used fully, it is possible to manually deallocate space from the table. The space thus released is available for use by other segments in the tablespace.

Syntax

Use the following command to deallocate unused space for a table:

```
ALTER TABLE [schema.]table
DEALLOCATE UNUSED [KEEP integer [ K|M ] ]
```

KEEP specifies the number of bytes above the high-water mark that should be retained.

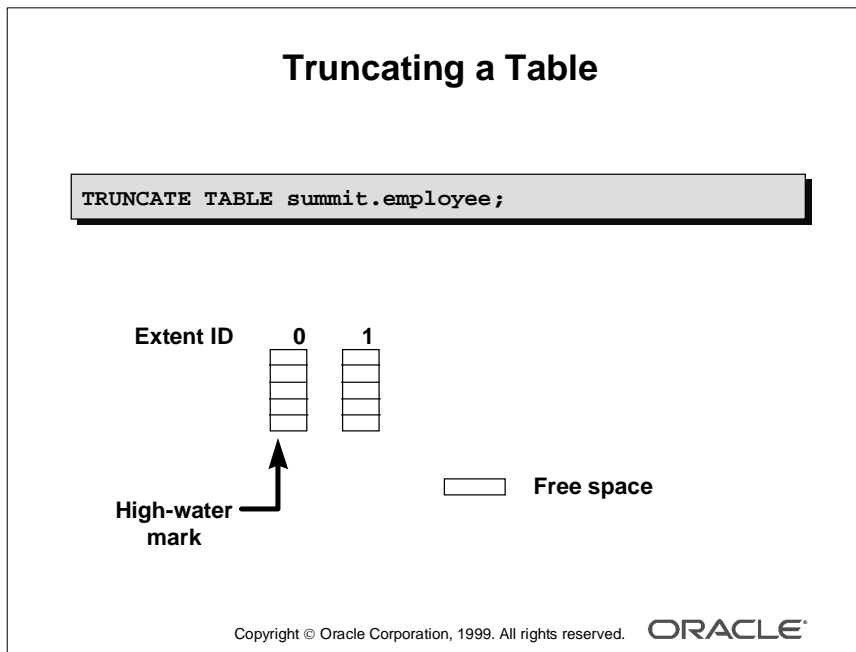
If the command is used without the **KEEP** clause, the Oracle server will deallocate all unused space above the high-water mark. If the high-water mark is at an extent less than the value of **MINEXTENTS**, the Oracle server will release extents above **MINEXTENTS**.

Consider the example in the slide. If **MINEXTENTS** for the table is four or lower, the Oracle server deallocates all unused blocks above the high-water mark, as shown. Notice that the fifth extent (with ID=4) now contains only five blocks. If **MINEXTENTS** is five for the table, the Oracle server does not have deallocated space from the fifth extent.

Syntax (continued)

Note

- Because deallocation of space using this command releases space unused within an extent, frequent use of this command may lead to fragmentation of the space in data files. To avoid this problem, set `MINIMUM EXTENT` for the tablespace.
- To release all the space below the high-water mark, even if the high-water mark is below `MINEXTENTS`, use `KEEP 0`.



Truncating a Table

Truncating a table deletes all rows in a table and releases used space. The example in the slide assumes that the current MINEXTENTS setting for the table is 2.

Syntax

```
TRUNCATE TABLE [schema.] table
[ {DROP | REUSE } STORAGE ]
```

The effects of using this command are as follows:

- All rows in the table are deleted.
- No rollback data is generated and the command commits implicitly because TRUNCATE TABLE is a DDL command.
- Corresponding indexes are also truncated.
- A table that is being referenced by a foreign key cannot be truncated.
- The delete triggers do not fire when this command is used.

Syntax (continued)

- If the DROP clause, which is the default, is used:
 - All extents except those specified by MINEXTENTS are deallocated.
 - The high-water mark is reset to point to the first block in the table.
 - The value of NEXT_EXTENT for the table is reset to the size of the extent with the lowest extent ID that is deallocated—that is, if MINEXTENTS is 2, NEXT_EXTENT size will be set to the size of the third extent of the table.
- You must specify the REUSE clause to retain all the space used by the table.
- The effect of REUSE or DROP cascades to the indexes.

Dropping a Table

```
DROP TABLE summit.department  
CASCADE CONSTRAINTS;
```

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE®

Dropping a Table

A table may be dropped if it is no longer needed or if it is to be reorganized.

Syntax

Use the following command to drop a table:

```
DROP TABLE [schema.] table  
[CASCADE CONSTRAINTS]
```

When a table is dropped, the extents used by the table are released. If they are contiguous, they may be coalesced either automatically or manually at a later stage.

Note: The CASCADE CONSTRAINTS option is necessary if the table is the parent table in a foreign key relationship. This option is discussed in detail in the lesson “Maintaining Data Integrity.”

How to Use Oracle Enterprise Manager to Drop a Table

- 1 Use Schema Manager.
- 2 Expand the Tables folder.
- 3 Expand the username (or schema).
- 4 Select the table.
- 5 Select Object—>Remove.
- 6 Select Yes in the dialog box.

Dropping a Column

Remove a column from a table

```
ALTER TABLE employee  
DROP COLUMN comments  
CASCADE CONSTRAINTS CHECKPOINT 1000;
```

- This removes the column length and data from each row, freeing space in the data block.
- Dropping a column in a large table takes a considerable amount of time.

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Removing a Column From a Table

Before Oracle8i, it was not possible to drop a column from a table. With Oracle8i, the Oracle server enables you to drop columns from rows in a table. Dropping columns cleans unused and potentially space-demanding columns without having to export or import data, and re-create indexes and constraints.

Dropping a column can take a significant amount of time because all the data for the column is deleted from the table.

Using a Checkpoint When Dropping a Column

Dropping a column can be time-consuming and require a large amount of rollback space. While dropping columns from large tables, checkpoints can be specified to minimize the use of rollback space. In the example in the slide, a checkpoint occurs every 1,000 rows. The table is marked INVALID until the operation completes. If the instance fails during the operation, the table remains INVALID on startup, and the operation will have to be completed.

Use the following statement to resume an interrupted drop operation:

```
SQL> ALTER TABLE orders  
DROP COLUMNS CONTINUE;
```

Using this statement generates an error if the table is in a VALID state.

Using the UNUSED Option

- **Mark a column as unused**

```
ALTER TABLE orders  
SET UNUSED COLUMN comments  
CASCADE CONSTRAINTS;
```

- **Drop unused columns**

```
ALTER TABLE orders  
DROP UNUSED COLUMNS CHECKPOINT 1000;
```

- **Continue to drop column operation**

```
ALTER TABLE orders  
DROP COLUMNS CONTINUE CHECKPOINT 1000;
```

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE®

Marking a Column as Unused

Instead of removing a column from a table, the column can be marked as unused and then removed later. This has the advantage of being relatively quick, as it does not reclaim the disk space because the data is not removed. Columns marked as unused can be removed at a later time from the table when there is less activity on the system.

Unused columns act as if they are not part of the table. Queries cannot see data from unused columns. In addition, the names and data types of those columns are not displayed when a DESCRIBE command is executed. A user can add a new column with the same name as an unused column.

An example of setting a column to unused before dropping it would be if you want to drop two columns in the same table. When you drop two columns, all rows in the table are updated twice; if you set the columns to unused and then drop the columns, the rows will only be updated once.

Identifying Tables with Unused Columns

To identify tables with unused columns, you can query the view `DBA_UNUSED_COL_TABS`. It obtains the names of tables that have unused columns and the number of columns that are marked unused in them. The following query shows that the table `ORDERS` owned by `SUMMIT` has one unused column:

```
SQL> select * from dba_unused_col_tabs;
```

OWNER	TABLE_NAME	COUNT

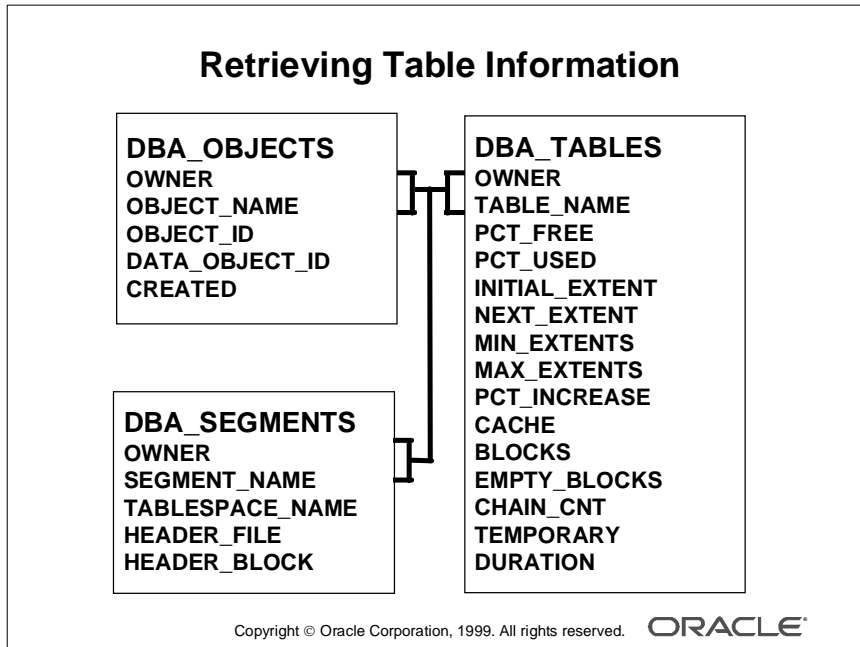
SUMMIT	ORDERS	1

Restrictions on Dropping a Column

You cannot do the following:

- Drop a column from an object type table
- Drop columns from nested tables
- Drop all columns in a table
- Drop a partitioning key column
- Drop a column from tables owned by `SYS`
- Drop a parent key column
- Drop a column from an index-organized table if the column is a primary key

Retrieving Table Information



Obtaining Data Dictionary Information

Information about tables can be obtained from the data dictionary. To obtain the data object number and the location of the table header for all tables owned by SUMMIT, use the following query:

```

SQL> SELECT t.table_name, o.data_object_id,
2 s.header_file, s.header_block
3 FROM dba_tables t, dba_objects o, dba_segments s
4 WHERE t.owner=o.owner
5 AND t.table_name=o.object_name
6 AND t.owner=s.owner
7 AND t.table_name=s.segment_name
8 AND t.owner='SUMMIT' ;
  
```

Obtaining Data Dictionary Information (continued)

TABLE_NAME	DATA_OBJECT_ID	HEADER_FILE	HEADER_BLOCK
<hr/>			
CUSTOMER	12743	2	902
DEPARTMENT	12745	2	912
EMPLOYEE	12748	2	927
IMAGE	12751	2	942
INVENTORY	12753	2	952
ITEM	12755	2	962
LONG_TEXT	12758	2	977
ORDERS	12760	2	987
PRODUCT	12762	2	997
REGION	12765	2	1012
TITLE	12768	2	1027
WAREHOUSE	12770	2	1037

12 rows selected.

Retrieving Extent Information

DBA_EXTENTS

- OWNER
- SEGMENT_NAME
- EXTENT_ID
- FILE_ID
- BLOCK_ID
- BLOCKS

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Distribution of Space Allocated

The number of extents, their location, and their size can be obtained by querying DBA_EXTENTS. The example below shows the number of extents and the total blocks used by a table in each file in the database:

```
SQL> SELECT file_id, COUNT(*) AS Extents, SUM(blocks) AS Blocks
2 FROM dba_extents
3 WHERE owner='SUMMIT'
4 AND segment_name='EMPLOYEE'
5 GROUP BY file_id;
```

FILE_ID	EXTENTS	BLOCKS
3	1	25

1 row selected.

DBMS_ROWID Package

Commonly used functions:

Function Name	Description
ROWID_CREATE	Creates a ROWID from individual components
ROWID_OBJECT	Returns the object identifier for a ROWID
ROWID_RELATIVE_FNO	Returns the relative file number for a ROWID
ROWID_BLOCK_NUMBER	Returns the block number for a ROWID
ROWID_ROW_NUMBER	Returns the row number for a ROWID
ROWID_TO_ABSOLUTE_FNO	Returns the absolute file number for a ROWID
ROWID_TO_EXTENDED	Converts a ROWID from restricted to extended
ROWID_TO_RESTRICTED	Converts a ROWID from extended to restricted

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Getting ROWID Information

The Oracle server provides a package called DBMS_ROWID, which is created from the `dbmsutil.sql` script, which in turn is called from `catproc.sql`.

The package provides several functions that can be used to convert between ROWID formats and to translate between ROWID and its individual components. Some examples of the uses of this package are shown in this section.

Getting ROWID Components

Use the following query to obtain the physical location of the rows in a table:

```
SQL> SELECT id, ROWID,
2 DBMS_ROWID.ROWID_OBJECT(ROWID) AS OBJECT,
3 DBMS_ROWID.ROWID_RELATIVE_FNO(ROWID) AS "RELATIVE FILE",
4 DBMS_ROWID.ROWID_BLOCK_NUMBER(ROWID) AS BLOCK
5 FROM summit.department;
```

Getting ROWID Components (continued)

ID	ROWID	OBJECT	RELATIVE	FILE	BLOCK
---	-----	-----	-----	-----	-----
10	AAADHJAACAAAAORAAA	12745		2	913
31	AAADHJAACAAAAORAAB	12745		2	913
32	AAADHJAACAAAAORAAC	12745		2	913
33	AAADHJAACAAAAORAAD	12745		2	913
34	AAADHJAACAAAAORAAE	12745		2	913
35	AAADHJAACAAAAORAAF	12745		2	913
41	AAADHJAACAAAAORAAG	12745		2	913
42	AAADHJAACAAAAORA AH	12745		2	913
43	AAADHJAACAAAAORAAI	12745		2	913
44	AAADHJAACAAAAORAAJ	12745		2	913
45	AAADHJAACAAAAORAAK	12745		2	913
50	AAADHJAACAAAAORAAL	12745		2	913

12 rows selected.

Finding the Absolute File Number

The following query can be used to get the absolute file numbers for rows in SUMMIT.DEPARTMENT:

```
SQL> SELECT id, ROWID,
  2 DBMS_ROWID.ROWID_TO_ABSOLUTE_FNO(ROWID, 'SUMMIT', 'DEPARTMENT')
AS "FILE"
  3 FROM summit.department;
```

ID	ROWID	FILE
---	-----	-----
10	AAADHJAACAAAAORAAA	2
31	AAADHJAACAAAAORAAB	2
32	AAADHJAACAAAAORAAC	2
33	AAADHJAACAAAAORAAD	2
34	AAADHJAACAAAAORAAE	2
35	AAADHJAACAAAAORAAF	2
41	AAADHJAACAAAAORAAG	2
42	AAADHJAACAAAAORA AH	2
43	AAADHJAACAAAAORAAI	2

Finding the Absolute File Number (continued)

44	AAADHJAACAAAAORAAJ	2
45	AAADHJAACAAAAORAAC	2
50	AAADHJAACAAAAORAAL	2

12 rows selected.

Summary

Summary

In this lesson, you should have learned how to:

- **Create a table with appropriate storage and block utilization parameters**
- **Control table storage**
- **Use the DBMS_ROWID package**

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

Quick Reference

Context	Reference
Initialization parameters	DB_BLOCK_SIZE
Dynamic initialization parameters	DB_FILE_MULTIBLOCK_READ_COUNT
Dynamic performance views	None
Data dictionary views	DBA_TABLES DBA_SEGMENTS DBA_OBJECTS DBA_EXTENTS
Commands	CREATE TABLE CREATE GLOBAL TEMPORARY TABLE ALTER TABLE ... STORAGE ALTER TABLE ... PCTFREE ... PCTUSED ALTER TABLE ... ALLOCATE EXTENT ALTER TABLE ... DEALLOCATE UNUSED TRUNCATE TABLE DROP TABLE ALTER TABLE ... DROP COLUMN ALTER TABLE ... SET UNUSED COLUMN ALTER TABLE ... DROP UNUSED COLUMNS ALTER TABLE ... DROP COLUMNS CONTINUE
Packaged procedures and functions	DBMS_SPACE.UNUSED_SPACE DBMS_ROWID

Managing Indexes

Objectives

Objectives

After completing this lesson, you should be able to do the following:

- **List the different types of indexes and their uses**
- **Create B-tree and bitmap indexes**
- **Reorganize indexes**
- **Drop indexes**
- **Get index information from the data dictionary**

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

Overview

Classification of Indexes

- **Logical**
 - **Single column or concatenated**
 - **Unique or nonunique**
 - **Function-based**
- **Physical**
 - **Partitioned or nonpartitioned**
 - **B-tree**
 - **Normal or reverse key**
 - **Bitmap**

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE®

Classification of Indexes

An index is a tree structure that allows direct access to a row in a table. Indexes can be classified based on their logical design or on their physical implementation. The logical classification groups indexes from an application perspective, while the physical classification is derived from the way the indexes are stored.

Single Column and Concatenated Indexes

A *single column index* has only one column in the index key—for example, an index on the employee number column of an employee table.

A *concatenated index*, also known as a composite index, is created on multiple columns in a table. Columns in a concatenated index do not need to be in the same order as the columns in the table, nor do they need to be adjacent—for example, an index on the department and job columns of an employee table.

The maximum number of columns in a composite key index is 32. However, the combined size of all the columns cannot exceed roughly one-third of the data block size.

Unique and Nonunique Indexes

A unique index guarantees that no two rows of a table have duplicate values in the column that defines the index. An index key in a unique index can point to only one row in the table.

In a nonunique index, a single key can have multiple rows associated with it.

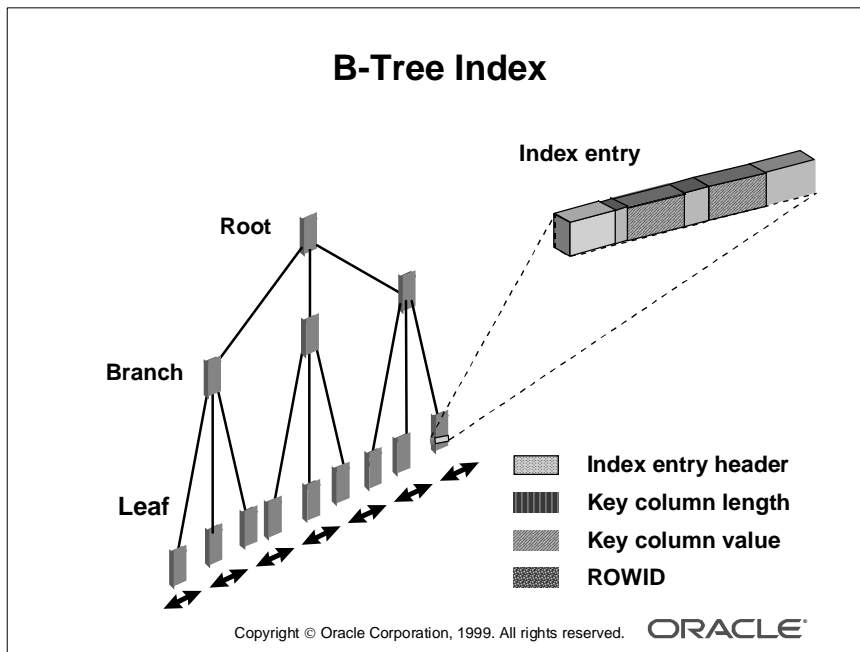
Function-Based Indexes

A function-based index is created when using functions or expressions that involve one or more columns in the table being indexed. A function-based index precomputes the value of the function or expression and stores it in the index. Function-based indexes can be created as either a B-tree or a bitmap index.

Partitioned and Nonpartitioned Indexes

Partitioned indexes are used for large tables to store index entries corresponding to an index in several segments. Partitioning allows an index to be spread across many tablespaces, decreasing contention for index lookup, and increasing manageability. Partitioned indexes are often used with partitioned tables to improve scalability and manageability. An index partition can be created for each table partition.

This lesson discusses the creation and maintenance of nonpartitioned B-tree and bitmap indexes.



How Indexes Are Stored

Although all the indexes use a B-tree structure, the term B-tree index is usually associated with an index that stores a list of ROWIDS for each key.

Structure of a B-Tree Index

At the top of the index is the root, which contains entries that point to the next level in the index. At the next level are branch blocks, which in turn point to blocks at the next level in the index. At the lowest level are the leaf nodes, which contain the index entries that point to rows in the table. The leaf blocks are doubly linked to facilitate scanning the index in an ascending as well as descending order of key values.

Format of Index Leaf Entries

An index entry is made up of the following components:

- An entry header, which stores number of columns and locking information
- Key column length-value pairs, which define the size of a column in the key followed by the value for the column (The number of such pairs is a maximum of the number of columns in the index.)
- ROWID of a row, which contains the key values

Index Leaf Entry Characteristics

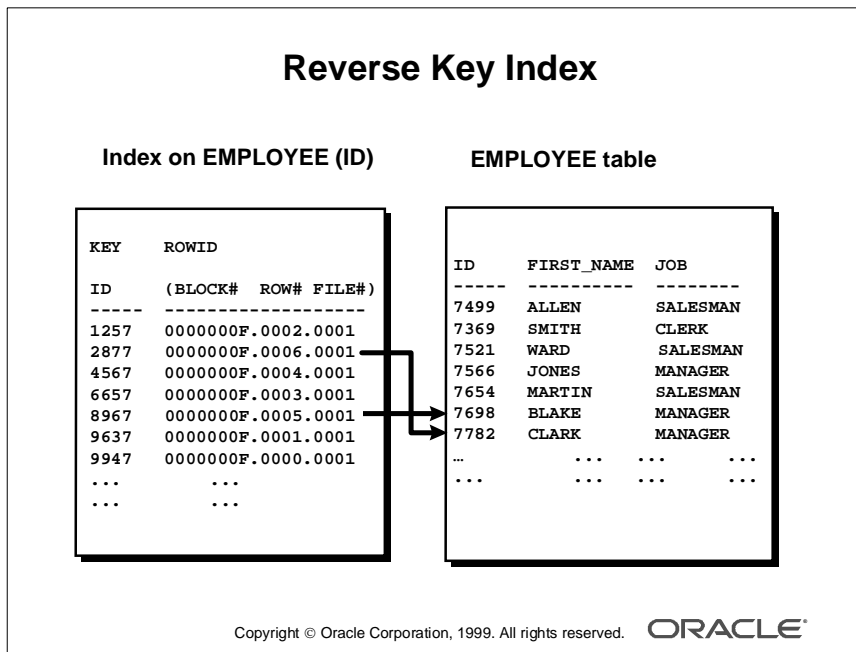
In a B-tree index on a nonpartitioned table:

- Key values are repeated if there are multiple rows that have the same key value.
- There is no index entry corresponding to a row that has all key columns that are NULL.
- Restricted ROWID is used to point to the rows of the table, since all rows belong to the same segment.

Effect of DML Operations on an Index

The Oracle server maintains all the indexes when DML operations are carried out on the table. Here is an explanation of the effect of a DML command on an index:

- Insert operations result in the insertion of an index entry in the appropriate block.
- Deleting a row results only in a logical deletion of the index entry. The space used by the deleted row is not available for new entries until all the entries in the block are deleted.
- Updates to the key columns result in a logical delete and an insert to the index. The PCTFREE setting has no effect on the index except at the time of creation. A new entry may be added to an index block even if it has less space than that specified by PCTFREE.



Reverse Key Indexes

In contrast to a regular B-tree index, a reverse key index reverses the bytes of each column indexed (except the ROWID) while keeping the column order. When inserting records on an ascending key, such as a system-generated employee number, I/O bottlenecks can occur on the index because all index updates occur at the same place in the index tree. Reverse key indexes spread the distribution of index updates across the index tree by reversing the data value of the index key.

For example, on insert of employee number 7698 in the table, a key value of 8967 is stored in the index. As the next employee number 7782 is entered, an index entry is made for 2877, thereby spreading the work load across multiple index blocks.

Reverse key indexes are useful only for queries that contain equality predicates. Because lexically adjacent keys are not stored next to each other in a reverse key index, range searches cannot be performed using such an index.

Creating Function-Based Indexes

- Dramatically improves query performance

```
CREATE INDEX summit.item_quantity_to_deliver_idx  
ON summit.item(quantity - quantity_shipped);
```

- Queries using expressions can use the index

```
SELECT ord_id, item_id  
FROM ITEM  
WHERE (quantity - quantity_shipped) > 0;
```

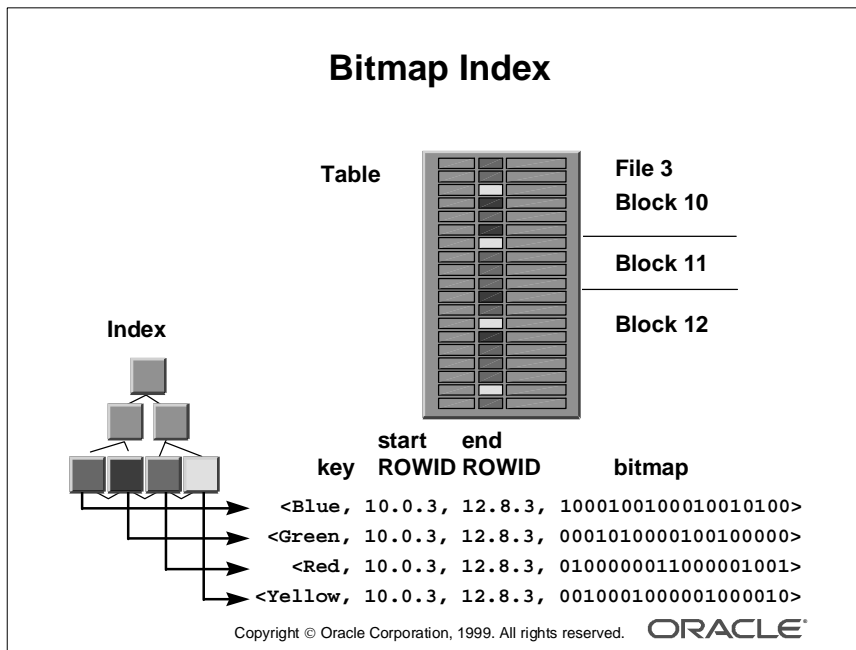
Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Function-Based Indexes

Function-based indexes provide an efficient mechanism for evaluating statements that contain functions in their WHERE clauses. A function-based index can be created to materialize computational-intensive expressions in the index, so that the Oracle server does not need to compute the value of the expression when processing SELECT and DELETE statements. When processing INSERT and UPDATE statements, however, the Oracle server must still evaluate the function to process the statement.

A function-based index can be created with either a B-tree or bitmap index structure.



Bitmap Indexes

Bitmap indexes are more advantageous than B-tree indexes in certain situations:

- When a table has millions of rows and the key columns have low cardinality—that is, there are very few distinct values for the column. For example, bitmap indexes may be preferable to B-tree indexes for the gender and marital status columns of a table containing passport records.
- When queries often use a combination of multiple WHERE conditions involving the OR operator.
- When there is read-only or low update activity on the key columns.

Structure of a Bitmap Index

A bitmap index is also organized as a B-tree, but the leaf node stores a bitmap for each key value instead of a list of ROWIDS. Each bit in the bitmap corresponds to a possible ROWID, and if the bit is set, it means that the row with the corresponding ROWID contains the key value.

As shown in the diagram, the leaf node of a bitmap index contains the following:

- An entry header, containing the number of columns and lock information
- Key values consisting of length and value pairs for each key column (In the example, the key consists of only one column, and the first entry has a key value of Blue.)

Structure of a Bitmap Index (continued)

- Start ROWID, which in the example contains a file number 3, a block number 10, and a row number 0
- End ROWID, which in the example includes a block number 12 and a row number 8
- A bitmap segment consisting of a string of bits (The bit is set when the corresponding row contains the key value and is unset when the row does not contain the key value. The Oracle server uses a patented compression technique to store bitmap segments.)

The start ROWID is the ROWID of the first row pointed to by the bitmap segment of the bitmap—that is, the first bit of the bitmap corresponds to that ROWID, the second bit of the bitmap corresponds to the next row in the block, and the end ROWID is a pointer to the last row in the table covered by the bitmap segment. Bitmap indexes use restricted ROWIDs.

Using a Bitmap Index

The B-tree is used to locate the leaf nodes that contain bitmap segments for a given value of the key. Start ROWID and the bitmap segments are used to locate the rows that contain the key value.

When changes are made to the key column in the table, bitmaps must be modified. This results in locking of the relevant bitmap segments. Because locks are acquired on the whole bitmap segment, a row that is covered by the bitmap cannot be updated by other transactions until the first transaction ends.

Comparing B-Tree and Bitmap Indexes

B-tree	Bitmap
Suitable for high-cardinality columns	Suitable for low-cardinality columns
Updates on keys relatively inexpensive	Updates to key columns very expensive
Inefficient for queries using OR predicates	Efficient for queries using OR predicates
Useful for OLTP	Useful for data warehousing

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Comparing B-Tree and Bitmap Indexes

Bitmap indexes are more compact than B-tree indexes when used with low-cardinality columns.

Updates to key columns in a bitmap index are more expensive because bitmaps use bitmap-segment-level locking, whereas in a B-tree index, locks are on entries corresponding to individual rows of the table.

Bitmap indexes can be used to perform operations such as Bitmap OR—that is, the Oracle server can use two bitmap segments to perform a bitwise OR and get a resulting bitmap. This allows efficient use of bitmaps in queries that use the OR predicate.

In summary, B-tree indexes may be more suitable in an OLTP environment for indexing dynamic tables, whereas bitmap indexes may be useful in data warehouse environments where complex queries are used on large, static tables.

Creating Indexes

Creating Normal B-Tree Indexes

```
CREATE INDEX summit.employee_last_name_idx
ON summit.employee(last_name)
PCTFREE 30
STORAGE(INITIAL 200K NEXT 200K
PCTINCREASE 0 MAXEXTENTS 50)
TABLESPACE indx;
```

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE

Creating Normal B-Tree Indexes

An index can be created either in the account of the user who owns the table or in a different account, although it is generally created in the same account as the table.

Syntax

Use the following command to create a B-tree index:

```
CREATE [ UNIQUE ] INDEX [schema.] index
ON [schema.] table
(column [ ASC | DESC ] [ , column [ASC | DESC ] ] ...)
[ TABLESPACE tablespace ]
[ PCTFREE integer ]
[ INITTRANS integer ]
[ MAXTRANS integer ]
[ storage-clause ]
[ LOGGING | NOLOGGING ]
[ NOSORT]
```

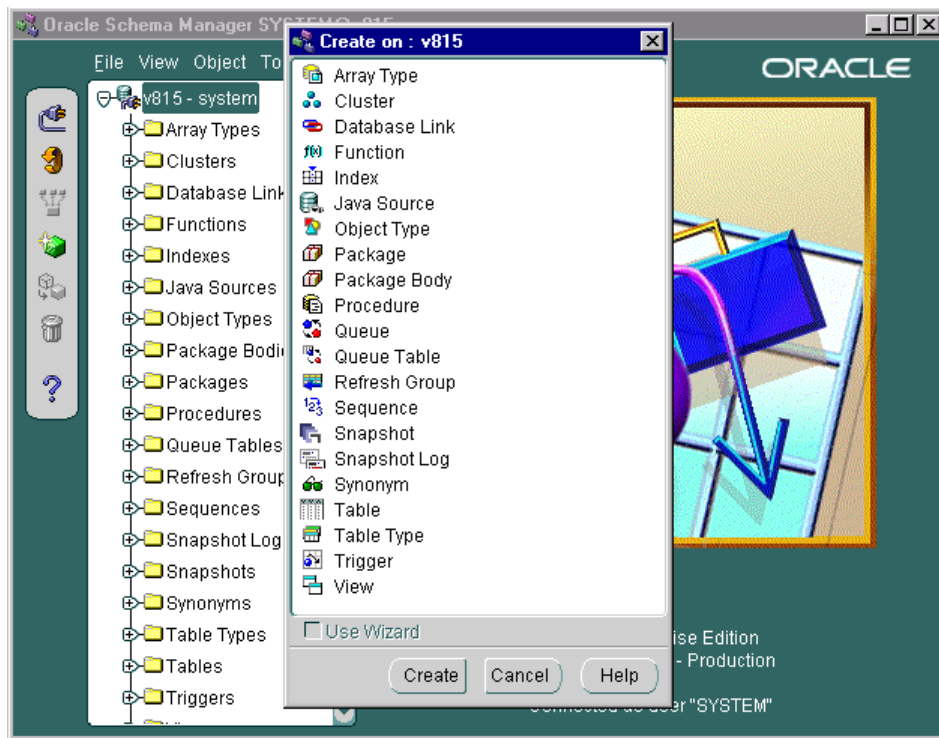
Syntax (continued)

where:	UNIQUE	is used to specify a unique index (Nonunique is the default.)
	schema	is the owner of the index/table
	index	is the name of the index
	table	is the name of the table
	column	is the name of the column
	ASC/ DESC	is provided for syntactic compatibility with other databases
	TABLESPACE	identifies the tablespace where the index will be created
	PCTFREE	is the amount of space reserved in each block (in percentage of total space minus the block header) at the time of creation for accommodating new index entries
	INITRANS	specifies the number of transaction entries preallocated in each block (The default and the minimum value is 2.)
	MAXTRANS	limits the number of transaction entries that can be allocated to each block (The default is 255.)
	Storage-clause	identifies the storage clause that determines how extents are allocated to the index
	LOGGING	specifies that the creation of the index and subsequent operations on the index are logged in the redo log file (This is the default.)
	NOLOGGING	specifies that the creation and certain types of data loads are not logged in the redo log file
	NOSORT	specifies that the rows are stored in the database in ascending order, and therefore, the Oracle server does not have to sort the rows while creating the index

Syntax (continued)

Note

- If `MINIMUM EXTENT` has been defined for the tablespace, the extent sizes for the index are rounded up to the next higher multiple of the `MINIMUM EXTENT` value.
- If the `LOGGING` or `NOLOGGING` clause is omitted, the logging attribute of the index defaults to the logging attribute of the tablespace in which it resides.
- `PCTUSED` cannot be specified for an index. Because index entries must be stored in the correct order, the user cannot control when an index block is used for inserts.
- If the `NOSORT` keyword is used when the data is not sorted on the key, the statement terminates with an error. This option is likely to fail if the table has had several DML operations on it.
- The Oracle server uses existing indexes to create a new index, if possible. This happens when the key for the new index corresponds to the leading part of the key of an existing index.



How to Use Oracle Enterprise Manager to Create an Index

- 1 Launch Schema Manager and connect directly to the database:
Start—>Programs—>Oracle - *EMV2 Home*—>DBA Management Pack
—>Schema Manager
- 2 Enter the login information, and click OK.
- 3 Select Object—>Create.
- 4 Select Index from the list of values and then click OK.
- 5 Enter General, Storage, and Options information in the property sheet.
- 6 Click Create.

While using Schema Manager, the user also has the option to let the tool automatically define the storage and block utilization parameters based on an estimate of the initial volume, the growth rate, the insert activity on the table, and the order in which rows are inserted.

Creating Indexes: Guidelines

- **Balance query and DML needs**
- **Place in separate tablespace**
- **Use uniform extent sizes: Multiples of five blocks or MINIMUM EXTENT size for tablespace**
- **Consider NOLOGGING for large indexes**
- **Set high PCTFREE if new key values are likely to be within the current range**

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Guidelines When Creating Indexes

Consider the following while creating an index:

- Indexes speed up query performance and slow down DML operations. Always minimize the number of indexes needed on volatile tables.
- Place indexes in a separate tablespace, not in a tablespace that has rollback segments, temporary segments, and tables.
- To minimize fragmentation, use a few standard extent sizes that are multiples of $5 \times \text{DB_BLOCK_SIZE}$.
- There could be significant performance gain for large indexes by avoiding redo generation. Consider using the NOLOGGING clause for creating large indexes.
- Because index entries are smaller compared to the rows they index, index blocks tend to have more entries per block. For this reason, INITRANS should generally be higher on indexes than on the corresponding tables.

Indexes and PCTFREE

The PCTFREE parameter for an index works differently from that of a table. This parameter is used only during creation of the index to reserve space for index entries that may need to be inserted into the same index block. Index entries are not updated. When a key column is updated, this involves a logical delete of the index entry and an insert.

Indexes and PCTFREE (continued)

Use a low PCTFREE for indexes on columns that are monotonically increasing, such as a system-generated invoice number. In these cases, new index entries are always appended to the existing entries and there is no need to insert a new entry between two existing index entries.

Where the value for an indexed column of an inserted row can take on any value—that is, the new value can fall within the current range of values—you should provide for a higher PCTFREE. An example of an index requiring a high PCTFREE is an index on the customer code column of an invoice table. In this case, it is useful to specify a value of PCTFREE as indicated by the following equation:

$$\frac{\text{Maximum number of rows} - \text{Initial number of rows}}{\text{Maximum number of rows}} \times 100$$

The maximum value can refer to a specific time period, such as a year.

Creating Reverse Key Indexes

```
CREATE UNIQUE INDEX summit.orders_id_idx
ON summit.orders(id) REVERSE
PCTFREE 30
STORAGE(INITIAL 200K NEXT 200K
PCTINCREASE 0 MAXEXTENTS 50)
TABLESPACE indx;
```

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Syntax

Use the following command to create a reverse key index:

```
CREATE [ UNIQUE ] INDEX [schema.] index
ON [schema.] table
(column [ ASC | DESC ] [ , column [ASC | DESC ] ] ...)
[ TABLESPACE tablespace ]
  [ PCTFREE integer ]
  [ INITTRANS integer ]
  [ MAXTRANS integer ]
  [ storage-clause ]
  [ LOGGING | NOLOGGING ]
  REVERSE
```

The command for creating a reverse key index is similar to that for regular indexes, except that the keyword **REVERSE** is used. Notice that the **NOSORT** keyword cannot be used for reverse key indexes.

Creating Bitmap Indexes

Use the parameter `CREATE_BITMAP_AREA_SIZE` to specify the amount of memory allocated for bitmap creation.

```
CREATE BITMAP INDEX orders_region_id_idx
ON summit.orders(region_id)
PCTFREE 30
STORAGE(INITIAL 200K NEXT 200K
PCTINCREASE 0 MAXEXTENTS 50)
TABLESPACE indx;
```

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE®

Syntax

Use the following command to create a bitmap index:

```
CREATE BITMAP INDEX [schema.] index
ON [schema.] table
(column [ ASC | DESC ] [ , column [ASC | DESC ] ] ...)
[ TABLESPACE tablespace ]
[ PCTFREE integer ]
[ INITTRANS integer ]
[ MAXTRANS integer ]
[ storage-clause ]
[ LOGGING | NOLOGGING ]
[ NOSORT ]
```

Notice that a bitmap index cannot be unique.

CREATE_BITMAP_AREA_SIZE

The initialization parameter `CREATE_BITMAP_AREA_SIZE` determines the amount of space that will be used for storing bitmap segments in memory. The default value is 8 MB. A larger value may lead to a faster index creation. If cardinality is very small, this value can be set to a small value. For example, if cardinality is only 2, then the value can be in the order of kilobytes rather than megabytes. As a general rule, for a higher cardinality, more memory is needed for optimal performance.

Create Index - system@v815

General | Partitions | Storage | Options

Name:

Schema:

Tablespace:

Index On: ☒ Table ☐ Cluster

Schema:

Table:

Table Columns	Order
ADDRESS	
CITY	
COMMENTS	
COUNTRY	
CREDIT_RATING	1
ID	
NAME	
PHONE	
REGION_ID	
SALES_REP_ID	

Options

☐ Unique ☒ Bitmap ☐ Unsorted ☐ Reverse ☐ No Logging

Create Cancel Show SQL Help

How to Use Oracle Enterprise Manager to Create a Bitmap Index

- 1 Launch Schema Manager and connect directly to the database:
Start—>Programs—>Oracle - *EMV2 Home*—>DBA Management Pack
—>Schema Manager
- 2 Enter the login information, and click OK.
- 3 Select Object—>Create.
- 4 Select Index from the list of values and then click OK.
- 5 Enter General, Storage, and Options information in the property sheet, and make sure that the Bitmap option is selected in the General tab.
- 6 Click Create.

Reorganizing Indexes

Changing Storage Parameters for Indexes

```
ALTER INDEX summit.employee_last_name_idx  
STORAGE(NEXT 400K  
MAXEXTENTS 100);
```

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE

Changing Storage Parameters for Indexes

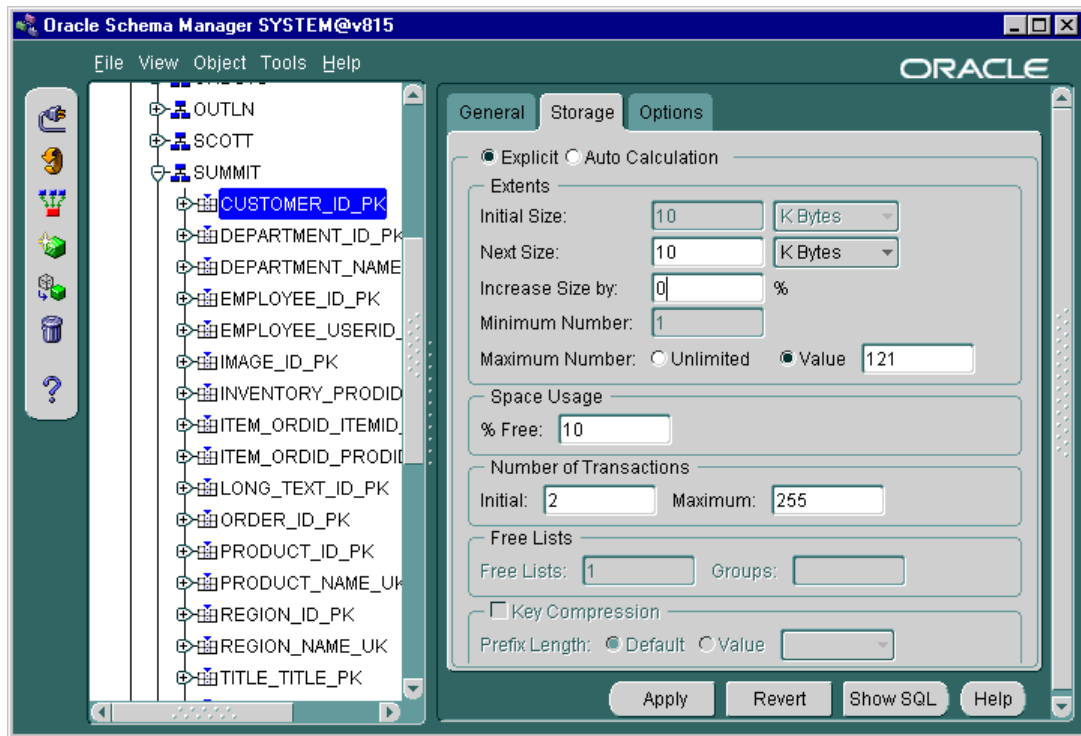
Some of the storage parameters and block utilization parameters can be modified by using the ALTER INDEX command.

Syntax

```
ALTER INDEX [schema.]index  
[ storage-clause ]  
[ INITTRANS integer ]  
[ MAXTRANS integer ]
```

The implications of changing the storage parameters for an index are the same as the implications of changing them for a table. A common use of this change is to increase the MAXEXTENTS for an index.

Block utilization parameters may be changed to guarantee higher levels of concurrency on an index block.



How to Use Oracle Enterprise Manager to Change Storage Parameters

- 1 Launch Schema Manager and connect directly to the database:
Start—>Programs—>Oracle - *EMV2 Home*—>DBA Management Pack
—>Schema Manager
- 2 Enter the login information, and click OK.
- 3 Expand the Indexes folder.
- 4 Expand the username (or schema).
- 5 Select the index.
- 6 Modify the values in the Storage tab of the property sheet. Note that minimum extents cannot be modified using this method.
- 7 Click Apply.

Allocating and Deallocating Index Space

```
ALTER INDEX summit.orders_region_id_idx  
ALLOCATE EXTENT (SIZE 200K  
DATAFILE '/DISK6/indx01.dbf');
```

```
ALTER INDEX summit.orders_id_idx  
DEALLOCATE UNUSED;
```

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Manual Allocation of Space to an Index

It may be necessary to add extents to an index before a period of high insert activity on a table. Adding extents prevents dynamic extension of indexes and the resulting degradation in performance.

Manual Deallocation of Space from an Index

Use the DEALLOCATE clause of the ALTER INDEX command to release unused space above the high-water mark in an index.

Syntax

Use the following command to allocate or deallocate index space:

```
ALTER INDEX [schema.]index  
{ALLOCATE EXTENT ([SIZE integer [K|M]]  
[ DATAFILE 'filename' ])  
| DEALLOCATE UNUSED [KEEP integer [ K|M ] ] }
```

Manual allocation and deallocation of space for an index follow the same rules as those that are used when using these commands against a table.

Note: Index space is deallocated when the table on which the index built is truncated. Truncating a table results in truncation of the associated index.

Rebuilding Indexes

Use the **ALTER INDEX** command to:

- **Move an index to a different tablespace**
- **Improve space utilization by removing deleted entries**
- **Change a reverse key index to a normal B-tree index and vice versa**

```
ALTER INDEX summit.orders_region_id_idx REBUILD  
TABLESPACE indx02;
```

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Rebuilding Indexes

Index rebuilds have the following characteristics:

- A new index is built using an existing indexes as the data source.
- Sorts are not needed when an index is built using an existing index, resulting in better performance.
- The old index is deleted after the new index is built. During the rebuild, sufficient space is needed to accommodate both the old and the new index in their respective tablespaces.
- The resulting index does not contain any deleted entries. Therefore, this index uses space more efficiently.
- Queries can continue to use the existing index while the new index is being built.

Possible Rebuild Situations

Rebuild an index in the following situations:

- The existing index needs to be moved to a different tablespace. This may be necessary if the index is in the same tablespace as the table or if objects need to be redistributed across disks.

Possible Rebuild Situations (continued)

- An index contains many deleted entries. This is a typical problem with sliding indexes, such as an index on the order number of an orders table, where completed orders are deleted and new orders with higher numbers are added to the table. If a few old orders are outstanding, there may be several index leaf blocks with all but a few deleted entries.
- An existing normal index needs to be converted into a reverse key index. This may be the case when migrating applications from an earlier release of the Oracle server.
- The table of the index has been moved to another tablespace using the ALTER TABLE ... MOVE TABLESPACE command.

Syntax

Use the following command to rebuild an index:

```
ALTER INDEX [schema.] index REBUILD
[ TABLESPACE tablespace ]
[ PCTFREE integer ]
[ INITTRANS integer ]
[ MAXTRANS integer ]
[ storage-clause ]
[ LOGGING | NOLOGGING ]
[ REVERSE | NOREVERSE ]
```

The ALTER INDEX ... REBUILD command cannot be used to change a bitmap index to B-tree and vice versa. The REVERSE or NOREVERSE keyword can only be specified for B-tree indexes.

Online Rebuild of Indexes

Rebuilding indexes can be done with minimal table locking.

```
ALTER INDEX summit.orders_id_idx REBUILD ONLINE;
```

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Rebuilding Indexes Online

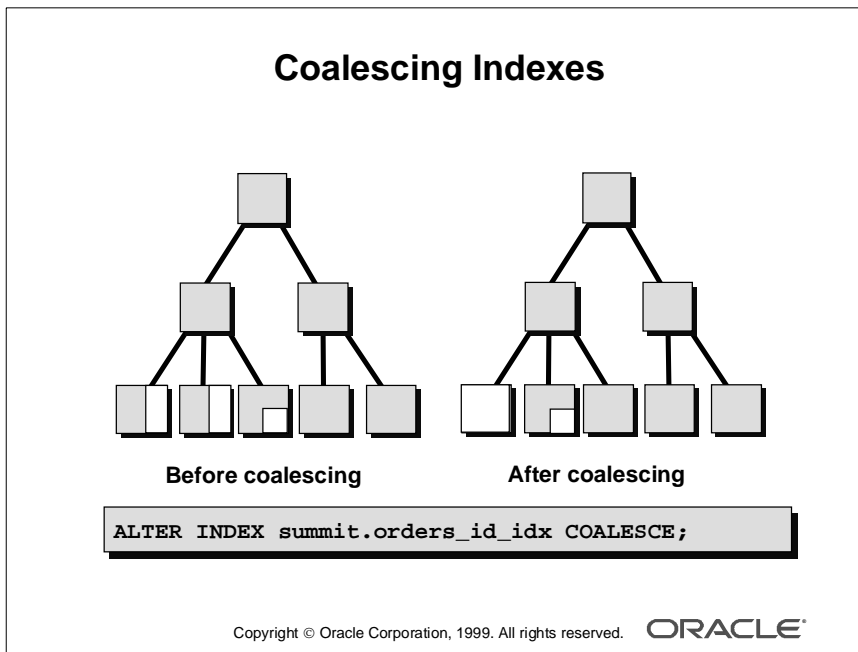
Building or rebuilding an index can be a time-consuming task, especially if the table is very large. Before Oracle8i, creating or rebuilding indexes required a lock on the table and prevented concurrent DML operations.

Oracle8i offers a method of creating or re-creating an index while allowing concurrent operations on the base table, but performing large DML operations during this procedure is not recommended.

Note: There are still DML locks, which means you cannot perform other DDL operations during an online index build.

Restrictions

- You cannot rebuild an index on a temporary table.
- You cannot rebuild an entire partitioned index. You must rebuild each partition or subpartition.
- You cannot also deallocate unused space.
- You cannot change the value of the PCTFREE parameter for the index as a whole.



Coalescing Indexes

When you encounter index fragmentation, you can rebuild or coalesce the index. Before you perform either task, you should consider the cost and benefits of each option and choose the one that works best for your situation.

In situations where you have B-tree index leaf blocks that can be freed up for reuse, you can merge those leaf blocks using the following SQL statement:

```
SQL> ALTER INDEX summit.orders_id_idx COALESCE;
```

Dropping Indexes

Dropping Indexes

- Drop and re-create an index before bulk loads.
- Drop indexes that are infrequently needed and build them when necessary.
- Drop and re-create invalid indexes.

```
DROP INDEX summit.department_name_idx;
```

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE

When Should Indexes Be Dropped?

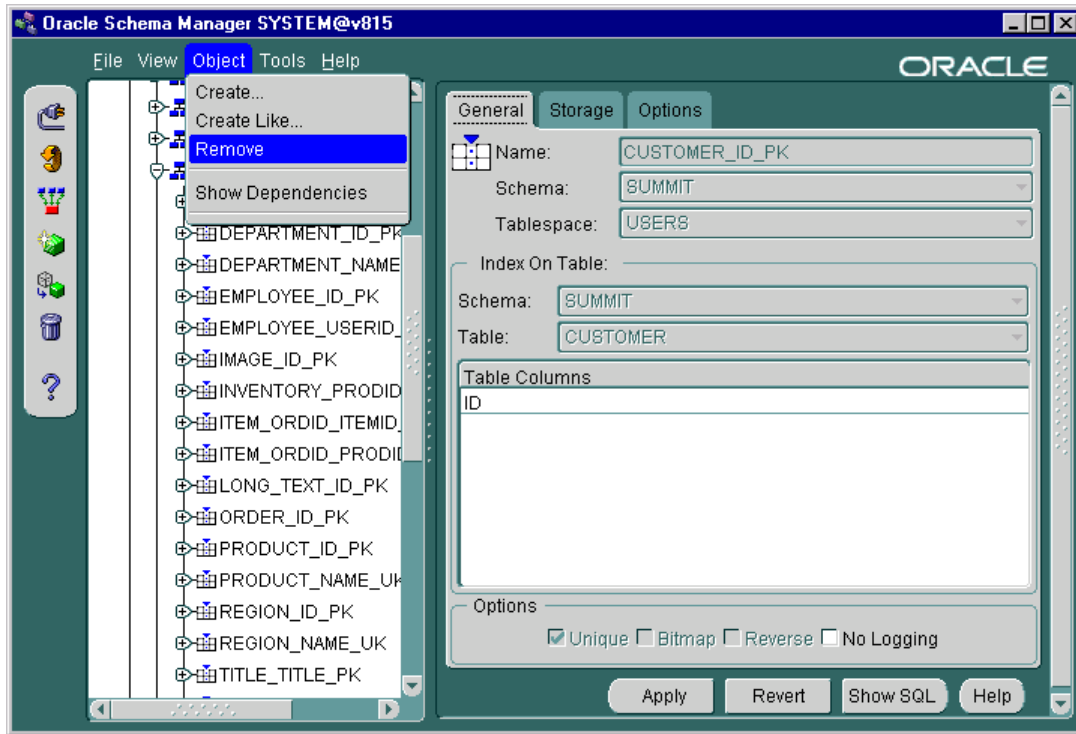
Indexes may need to be dropped in the following scenarios:

- An index that is no longer needed by applications can be removed.
- An index may be dropped prior to performing bulk loads. Dropping indexes prior to large data loads and re-creating them after the load:
 - Improves performance of the load
 - Uses index space more efficiently
- Indexes that are used only periodically do not need to be maintained unnecessarily, especially if they are based on volatile tables. This is generally the case in an OLTP system, where ad hoc queries are generated at year-end or quarter-end to gather information for review meetings.
- An index may be marked INVALID when there is an instance failure during certain types of operations such as loading. In this case, the index needs to be dropped and re-created.
- The index is corrupt.

Syntax

Use the following command to drop an index:

```
DROP INDEX [schema.] index
```

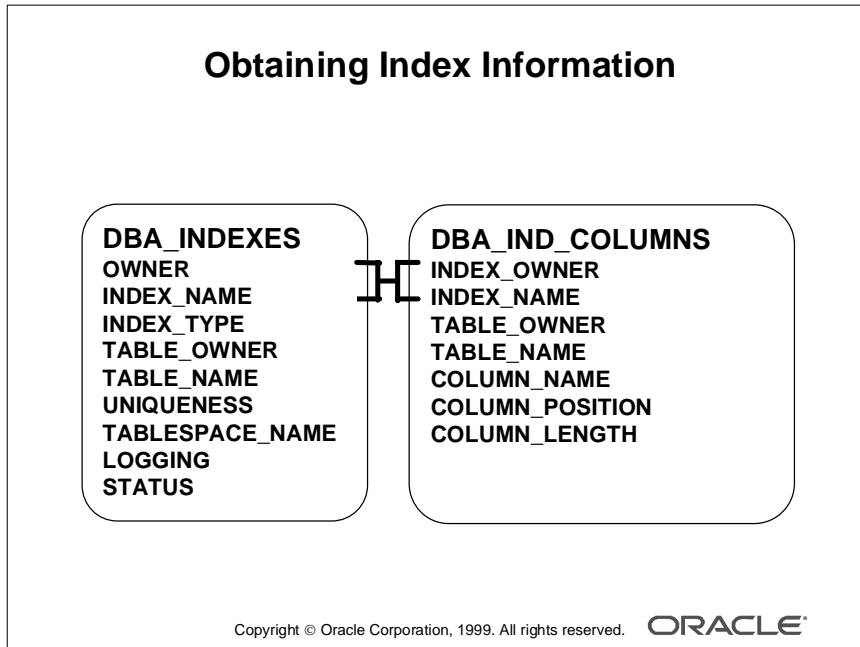


How to Use Oracle Enterprise Manager to Drop an Index

- 1 Launch Schema Manager and connect directly to the database:
Start—>Programs—>Oracle - *EMV2 Home*—>DBA Management Pack
—>Schema Manager
- 2 Enter the login information, and click OK.
- 3 Expand the Indexes folder.
- 4 Expand the username (or schema).
- 5 Select the index.
- 6 Select Object—>Remove.
- 7 Select Yes in the dialog box.

Note: An index cannot be dropped if it is used to implement an integrity constraint that is enabled. Constraints are discussed in the lesson “Maintaining Data Integrity.”

Obtaining Index Information



Views to Obtain Index Information

The data dictionary views `DBA_INDEXES` and `DBA_IND_COLUMNS` give the information on the indexes and the columns indexed.

Checking Indexes and Their Validity

Use the following command to verify the name, type, and status of the indexes owned by the user `SUMMIT`:

```
SQL> SELECT index_name, tablespace_name, index_type,
2 uniqueness, status
3 FROM dba_indexes
4 WHERE owner='SUMMIT';
```

INDEX_NAME	TABLESPACE_NAME	INDEX_TYPE	UNIQUENES	STATUS
-----	-----	-----	-----	-----
EMPLOYEE_LAST_..	INDX	NORMAL	NONUNIQUE	VALID
ORDERS_ID_IDX	INDX	NORMAL	UNIQUE	VALID
ORDERS_REGION_..	INDX02	BITMAP	NONUNIQUE	VALID

3 rows selected.

Checking Indexes and Their Validity (continued)

The column INDEX_TYPE indicates whether the index is bitmap or normal.

Use the following query to list the names of all reverse key indexes:

```
SQL> SELECT o.object_name
      2 FROM dba_objects o
      3 WHERE owner='SUMMIT'
      4 AND o.object_id IN (SELECT i.obj#
      5 FROM ind$ i
      6 WHERE BITAND(i.property,4) = 4);
OBJECT_NAME
-----
ORDERS_ID_IDX
1 row selected.
```

Finding Columns in an Index

The following query lists all the indexes owned by the user SUMMIT and shows the tables and columns on which the indexes are built:

```
SQL> SELECT index_name, table_owner, table_name, column_name
      2 FROM dba_ind_columns
      3 WHERE index_owner = 'SUMMIT'
      4 ORDER BY index_name, column_position;
INDEX_NAME          TABLE_OWNER    TABLE_NAME     COLUMN_NAME
-----
EMPLOYEE_LAST_NAME.. SUMMIT          EMPLOYEE        LAST_NAME
ORDERS_ID_IDX       SUMMIT          ORDERS          ID
ORDERS_REGION_ID..  SUMMIT          ORDERS          REGION_ID
3 rows selected.
```

Summary

Summary

In this lesson, you should have learned how to:

- **Create different types of indexes**
- **Reorganize indexes**
- **Drop indexes**
- **Get index information from the data dictionary**

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

Quick Reference

Context	Reference
Initialization parameters	CREATE_BITMAP_AREA_SIZE
Dynamic performance views	None
Data dictionary tables/views	DBA_INDEXES DBA_IND_COLUMNS DBA_OBJECTS IND\$ INDEX_STATS
Commands	CREATE INDEX CREATE UNIQUE INDEX CREATE BITMAP INDEX CREATE INDEX ... REVERSE ALTER INDEX ... STORAGE ALTER INDEX ... INITRANS ... MAXTRANS ALTER INDEX ... ALLOCATE EXTENT ALTER INDEX ... DEALLOCATE UNUSED ALTER INDEX REBUILD ALTER INDEX REBUILD ... REVERSE ALTER INDEX REBUILD ... NOREVERSE ANALYZE INDEX ... VALIDATE STRUCTURE DROP INDEX
Packaged procedures and functions	None

Maintaining Data Integrity

Objectives

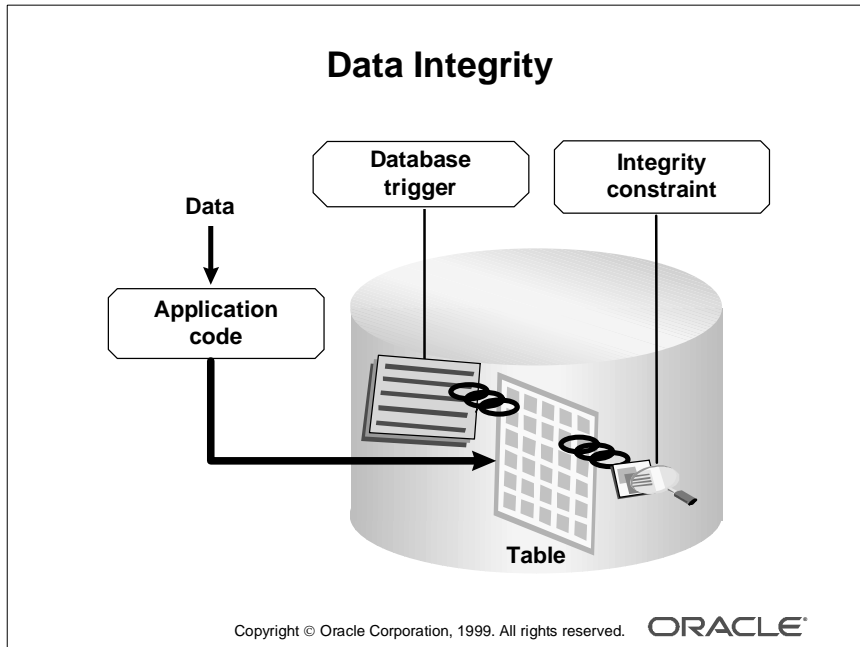
Objectives

After completing this lesson, you should be able to do the following:

- **Implement data integrity constraints**
- **Maintain integrity constraints**
- **Obtain constraint information from the data dictionary**

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

Overview



Methods to Guarantee Data Integrity

Data integrity guarantees that data in a database adheres to business rules. There are three primary ways in which data integrity can be maintained:

- Application code
- Database triggers
- Declarative integrity constraints

Mapping the business rules using one of the three methods is a design decision made by the designer. The database administrator is primarily concerned with implementing the methods chosen by the designer and balancing the performance needs against integrity requirements.

Application code may be implemented either as stored procedures within the database or as applications running on the client. This lesson focuses on the use of database triggers and integrity constraints.

Database Triggers

Database triggers are PL/SQL programs that are executed when a certain event such as an insert or an update of a column occurs on a table. Triggers can be enabled or disabled—that is, they can be set to execute when the event occurs, or they can be set not to execute even though they are defined. Database triggers are usually created only to enforce a complex business rule that cannot be defined as an integrity constraint.

Note: Database triggers are covered in other Oracle courses.

Integrity Constraints

Integrity constraints are the preferred mechanism for enforcing business rules because they:

- Provide improved performance
- Are easy to declare and modify—because they do not require extensive coding
- Centralize rules
- Are flexible (enabled or disabled)
- Are fully documented in the data dictionary

The following sections explain the behavior of integrity constraints and discuss how they are implemented by the Oracle server.

Integrity Constraints

Types of Constraints

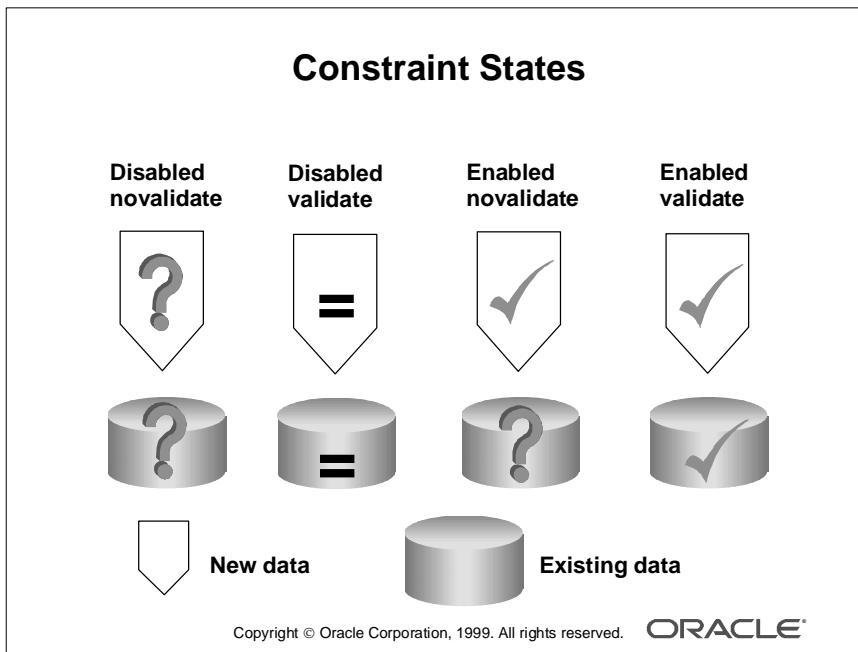
Constraint	Description
NOT NULL	Specifies that a column cannot contain null values
UNIQUE	Designates a column or combination of columns as unique
PRIMARY KEY	Designates a column or combination of columns as the table's primary key
FOREIGN KEY	Designates a column or combination of columns as the foreign key in a referential integrity constraint
CHECK	Specifies a condition that each row of the table must satisfy

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE

Types of Constraints

As shown in the slide, there are five types of declarative integrity constraints.

Although the NOT NULL and CHECK constraints do not directly require DBA attention, the primary key, unique, and foreign key constraints must be managed to ensure high availability and acceptable performance levels.



Constraint States

An integrity constraint can be in one of the following states:

- Disabled novalidate
- Disabled validate
- Enabled novalidate or enforced
- Enabled validate

Disabled Novalidate A constraint that is disabled novalidate is not checked, even though the constraint definition is still stored in the data dictionary. Data in the table, as well as new data that is entered or updated, may not conform to the rules defined by the constraint. This is the normal state of operation of a constraint for online transaction processing.

Disabled Validate If a constraint is in this state, then any modification of the constrained columns is not allowed. In addition, the index on the constraint is dropped and the constraint is disabled.

For a unique constraint, the disable validate state enables you to load data efficiently from a nonpartitioned table into a partitioned table using the EXCHANGE PARTITION option of the ALTER TABLE command.

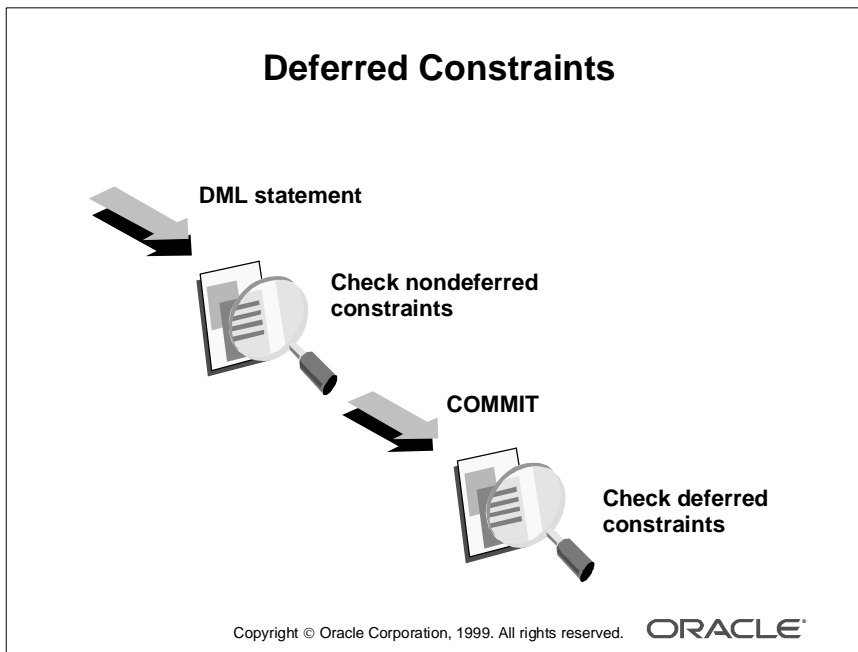
Note: Partitioned tables are not covered in this course.

Constraint States (continued)

Enabled Novalidate (Enforced) If a constraint is enabled novalidate, new data that violates the constraint cannot be entered. However, the table may contain data that is invalid—that is, data that violates the constraint. This is usually an intermediate stage that ensures that all new data is checked before being accepted into the table.

Enabled Validate If a constraint is in this state, then all data in the table is guaranteed to adhere to the constraint. In addition, this state prevents any invalid data from being entered. This is the normal state of operation of a constraint for online transaction processing.

When a constraint changes to enabled validate from a disabled state, the table is locked and all data in the table is checked for conformity. This may cause DML operations such as a data load to wait, so it is advisable to move first from a disabled state to enable novalidate, and then to enable validate.



Deferred Constraints

You can control the point in a transaction at which a constraint is checked can be controlled by defining the constraint appropriately.

Nondeferred or Immediate Constraints

Nondeferred constraints, also known as immediate constraints, are enforced at the end of every DML statement. A constraint violation causes the statement to be rolled back. If a constraint causes an action such as delete cascade, the action is taken as part of the statement that caused it.

A constraint that is defined as nondeferred cannot be modified to be enforced at the end of a transaction.

Deferred Constraints

Deferred constraints are constraints that are checked only when a transaction commits. If any constraint violations are detected at commit time, the entire transaction is rolled back. These constraints are most useful when both the parent and child rows in a foreign key relationship are entered at the same time, as in the case of an order entry system, where the order and the items in the order are entered at the same time.

Deferred Constraints (continued)

For a constraint to be deferred, it must be defined as a deferrable constraint at the time of creation. A constraint that is defined as deferrable can be specified as one of the following:

- *Initially immediate* specifies that by default it should function as an immediate constraint, unless explicitly set otherwise.
- *Initially deferred* specifies that by default the constraint should only be enforced at the end of the transaction.

Defining Constraints as Immediate or Deferred

Use the command **ALTER SESSION** or **SET CONSTRAINTS** to set the mode for your constraint.

```
ALTER SESSION
SET CONSTRAINT[S] =
{IMMEDIATE|DEFERRED|DEFAULT};
```

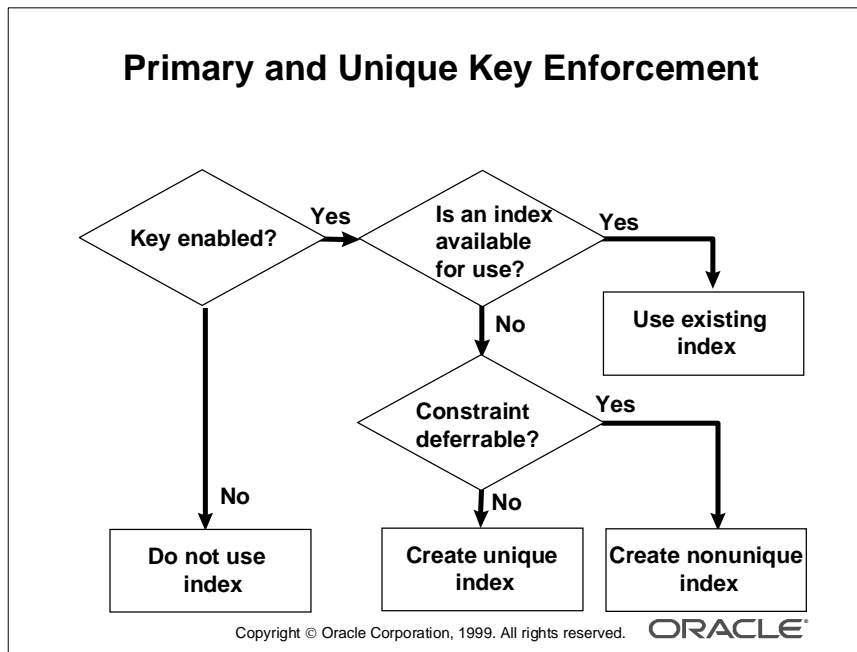
Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

Changing the Enforcement of Constraints

Although the default mode of enforcement of a deferrable constraint is specified and stored in the data dictionary, applications can modify the constraint to work as either deferred or immediate. This is accomplished by using either an **ALTER SESSION** command or a **SET CONSTRAINT** command:

```
ALTER SESSION
SET CONSTRAINT[S] =
{IMMEDIATE|DEFERRED|DEFAULT}

SET CONSTRAINT[S]
{constraint [, constraint ]...
|ALL      }
{IMMEDIATE|DEFERRED}
```



How to Enforce Primary and Unique Key Constraints

Primary and unique keys are enforced using indexes. You can control the location and type of index that is used for enforcing these constraints.

The Oracle server uses the following procedure to implement unique and primary key constraints:

- If the constraint is disabled, no indexes are needed.
- If the constraint is enabled and the columns in the constraint form the leading part of an index, the index will be used to enforce the constraint.
- If the constraint is enabled and there is no index that uses the constraint columns as a leading part of the index, an index with the same name as the constraint is created using the following rules:
 - If the key is deferrable, a nonunique index on the key column is created.
 - If the key is nondeferrable, a unique index is created.

Foreign Key Considerations

Desired Action	Appropriate Solution
Drop parent table	Cascade constraints
Truncate parent table	Disable or drop foreign key
Drop tablespace containing parent table	Use CASCADE CONSTRAINTS clause
Avoid locks on child table while performing DML on parent table	Create index on foreign key
Perform DML on child table	Ensure tablespace containing parent key index online

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

Considerations When Using Foreign Key Constraints

You need to consider several factors in maintaining tables that are in a foreign key relationship.

DDL Involving Parent Table

- The foreign key must be dropped before dropping the parent table. Use the following command to perform both actions using a single statement:
`DROP TABLE table CASCADE CONSTRAINTS`
- The parent table cannot be truncated without dropping or disabling the foreign key.
- The foreign key must be dropped before the tablespace containing the parent is dropped. The following command can be used to achieve this:
`DROP TABLESPACE tablespace INCLUDING CONTENTS
CASCADE CONSTRAINTS`

DML on Tables in a Foreign Key Relationship

If the DELETE CASCADE option is not used when rows are deleted from the parent table, the Oracle server needs to ensure that there are no rows in the child table with the corresponding foreign key. Similarly, an update to the parent key is permitted only when there are no child rows with the old key value. If there is no index on the foreign key on the child table, the Oracle server locks the child table and prevents changes to ensure referential integrity. If there is an index on the table, the referential integrity is maintained by locking the index entries and avoiding more restrictive locks on the child table. If both tables need to be updated concurrently from different transactions, create an index on the foreign key columns.

When data is inserted into or the foreign key column is updated in the child table, the Oracle server checks the index on the parent table that is used for enforcing the referenced key. Therefore, the operation succeeds only if the tablespace containing the index is online. Note that the tablespace containing the parent table does not need to be online to perform DML operations on the child table.

Note: Besides the advantage mentioned above, creating indexes on foreign key columns is recommended because it has other performance advantages.

Implementing Constraints

Defining Constraints While Creating a Table

```
CREATE TABLE summit.employee(  
  id NUMBER(7)  
    CONSTRAINT employee_id_pk PRIMARY KEY  
    DEFERRABLE  
    USING INDEX  
    STORAGE(INITIAL 100K NEXT 100K)  
    TABLESPACE indx,  
  last_name VARCHAR2(25)  
    CONSTRAINT employee_last_name_nn NOT NULL,  
  dept_id NUMBER(7))  
  TABLESPACE data;
```

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE®

Defining Constraints While Creating a Table

A constraint can be defined either when a table is created or when a table is altered to add the constraint.

Syntax: In-Line Constraint

At the time the table is created, the constraint can be created in-line using the following syntax to define the column:

```
column datatype [CONSTRAINT constraint]  
  in_line_constraint  
  [defer_spec]
```

Syntax: In-Line Constraint (continued)

```

in_line_constraint ::=
{[NOT] NULL
|PRIMARY KEY [USING INDEX index_clause]
|UNIQUE      [USING INDEX index_clause]
|REFERENCES [schema.]table [(column)]
              [ON DELETE CASCADE]
|CHECK      (condition)
}
defer_spec ::=
[NOT DEFERRABLE|DEFERRABLE [INITIALLY {IMMEDIATE|DEFERRED}]]
]
[DISABLE|ENABLE [VALIDATE|NOVALIDATE]]

```

where:	CONSTRAINT	identifies the integrity constraint by the name <i>constraint</i> stored in data dictionary
	USING INDEX	specifies that the parameters defined in the <i>index-clause</i> should be used for the index the Oracle server uses, to enforce a unique or primary key constraint (The name of the index is the same as the name of the constraint.)
	DEFERRABLE	indicates that constraint checking can be deferred until the end of the transaction by using the SET CONSTRAINT(S) command
	NOT DEFERRABLE	indicates that this constraint is checked at the end of each DML statement (A NOT DEFERRABLE constraint cannot be deferred by sessions or transactions. NOT DEFERRABLE is the default.)

Syntax: In-Line Constraint (continued)

INITIALLY IMMEDIATE

indicates that at the start of every transaction, the default is to check this constraint at the end of every DML statement (If no INITIALLY clause is specified, INITIALLY IMMEDIATE is the default.)

INITIALLY DEFERRED

implies that this constraint is DEFERRABLE and specifies that, by default, the constraint is checked only at the end of each transaction

DISABLE

disables the integrity constraint (If an integrity constraint is disabled, the Oracle server does not enforce it.)

Syntax: Out-of-Line Constraint

The constraint can also be created out-of-line, using the following syntax:

```
[CONSTRAINT constraint]
    out_of_line_constraint

    out_of_line_constraint ::=
    {PRIMARY KEY (column [, column ]... )
        [USING INDEX index_clause]
    |UNIQUE      (column [, column ]... )
        [USING INDEX index_clause]
    |FOREIGN KEY (column [, column ]... )
        REFERENCES [schema.]table [(column [, column ]... )]
        [ON DELETE CASCADE]
    |CHECK      (condition)
    }
    [defer_spec]
```

Note

- It is a good practice to adopt a standard naming convention for constraints. This is especially true with CHECK constraints because the same constraint can be created several times with different names.
- Out-of-line constraints are needed in the following cases:
 - When a constraint names two or more columns
 - When a table is altered to add any constraint other than the NOT NULL constraint

Defining Constraints After Creating a Table: Example

```
ALTER TABLE summit.employee
ADD(CONSTRAINT employee_dept_id_fk FOREIGN KEY(dept_id)
REFERENCES summit.department(id)
DEFERRABLE INITIALLY DEFERRED);
```

Note: The EXCEPTIONS clause, discussed under “Enabling Constraints” later in this lesson, can be used to identify rows violating a constraint that is added using the ALTER TABLE command.

Guidelines for Defining Constraints

- **Primary and unique constraints:**
 - **Place indexes in a separate tablespace**
 - **Use nonunique indexes if bulk loads are frequent**
- **Self-referencing foreign keys:**
 - **Define or enable foreign keys after initial load**
 - **Defer constraint checking**

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Guidelines for Defining Constraints

The following guidelines are useful when defining constraints:

- Place indexes used for enforcing primary key and unique constraints in a tablespace different from that of the table. This can be done either by specifying the USING INDEX clause or by creating the table, creating the index, and altering the table to add or enable the constraint.
- If data is frequently loaded in bulk into a table, it is preferable to disable the constraints, perform the load, and then enable the constraints. If a unique index is used for enforcing a primary key or unique constraint, this index needs to be dropped when the constraint is disabled. Performance can be enhanced by using a nonunique index for enforcement of primary key or unique constraints in such situations. This can be done either by creating the key as deferrable or by creating the index before defining or enabling the key.
- If a table contains a self-referencing foreign key, use one of the following methods to load data:
 - Define or enable the foreign key after the initial load
 - Define the constraint as a deferrable constraint

The second method is useful if data loads are done frequently.

Maintaining Constraints

Enabling Constraints

Enable
NOVALIDATE

- No locks on table
- Primary and unique keys must use nonunique indexes

```
ALTER TABLE summit.department
ENABLE NOVALIDATE CONSTRAINT dept_pk;
```

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

Enabling Constraints

A constraint that is currently disabled can be enabled in one of the two ways: enable NOVALIDATE or enable VALIDATE

Enable NOVALIDATE

Enabling a constraint novalidate is much faster than enabling a constraint validate because existing data is not checked for constraint violation if the constraint is deferrable. If this option is used for enabling a constraint, no locks are required on the table. This method is appropriate where there is a lot of DML activity on a table, as in the case of an OLTP environment.

Syntax

The following command can be used to enable a constraint novalidate:

```
ALTER TABLE [ schema. ] table
ENABLE NOVALIDATE {CONSTRAINT constraint
                    | PRIMARY KEY
                    | UNIQUE ( column [, column ] ... ) }
[ USING INDEX index_clause ]
```

Restrictions

The USING INDEX clause is applicable only for primary key or unique constraints that were created as deferrable, and one of the following is true:

- The constraints were created disabled.
- The constraints were disabled and the index dropped.

However, if the index needs to be created, using this method of enabling a constraint does not offer any significant benefit over enabling validate because the Oracle server locks the table to build the index.

Note: Disabling constraints is covered in the course *Introduction to SQL and PL/SQL*.

Enabling Constraints

Enable
VALIDATE

- Locks table
- Can use unique or nonunique indexes
- Needs valid table data

```
ALTER TABLE summit.employee  
ENABLE VALIDATE CONSTRAINT emp_dept_fk;
```

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

Enable VALIDATE

Enabling a constraint validate checks existing data for constraint violation. This is the default when a constraint is enabled. If executed when the constraint is disabled, it has the following effects:

- The table is locked and changes to the table are prevented until validation of existing data is complete.
- The Oracle server creates an index if one does not exist on the index columns. It creates a unique index while enabling a primary key or unique constraint that is nondeferrable. A nonunique index is built for a deferrable primary key or a unique constraint.

If this command is executed when a constraint is enforced, it does not require any table locks during validation. The enforced constraint guarantees that no violations are introduced during validation. This has the following advantages:

- All constraints are enabled concurrently.
- Each constraint is internally parallelized.
- Concurrent activity on the table is permitted.

Syntax

The following command is used to enable a constraint validate:

```
ALTER TABLE [ schema. ] table
ENABLE [ VALIDATE ] {CONSTRAINT constraint
    | PRIMARY KEY
    | UNIQUE ( column [, column ] ... ) }
[ USING INDEX index_clause ]
[ EXCEPTIONS INTO [ schema. ] table ]
```

Note

- The VALIDATE option is the default and does not need to be specified when enabling a constraint that is disabled.
- If data in the table violates the constraint, the statement is rolled back and the constraint remains disabled.
- The use of the EXCEPTIONS clause is discussed in the following section.

Using the EXCEPTIONS Table

1. Create EXCEPTIONS table (utlexcpt.sql).
2. Execute ALTER TABLE with EXCEPTIONS clause.
3. Use subquery on EXCEPTIONS to locate rows with invalid data.
4. Rectify the errors.
5. Reexecute ALTER TABLE to enable the constraint.

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

How to Identify Row Violation

The EXCEPTIONS clause helps to identify any row that violates a constraint that is being enabled. Use the following procedure to detect constraint violations, rectify them, and reenable a constraint:

- 1 If not already created, run the utlexcpt.sql script in the administration directory to create the exceptions table:

```
SQL> @?/rdbms/admin/utlexcpt
```

```
Statement processed.
```

```
SQL> DESCRIBE exceptions
```

Name	Null?	Type
ROW_ID		UNDEFINED
OWNER		VARCHAR2 (30)
TABLE_NAME		VARCHAR2 (30)
CONSTRAINT		VARCHAR2 (30)

On Windows NT, this script is located in the
%ORACLE_HOME%\RDBMS\ADMIN directory.

How to Identify Row Violation (continued)

2 Execute the ALTER TABLE command using the EXCEPTIONS clause:

```
SQL> ALTER TABLE summit.employee
      2  ENABLE VALIDATE CONSTRAINT employee_dept_id_fk
      3  EXCEPTIONS INTO system.exceptions;
ALTER TABLE summit.employee
*
ORA-02298: cannot enable (summit.EMP_DEPT_FK) - parent keys not found
```

If the EXCEPTIONS table is not qualified with the name of the owner, it must belong to the owner of the table being altered.

Rows are inserted into the EXCEPTIONS table. If you are rerunning the command, truncate the EXCEPTIONS table to remove all existing rows.

3 Identify invalid data by using a subquery on the EXCEPTIONS table:

```
SQL> SELECT rowid, id, last_name, dept_id
      2  FROM summit.employee
      3  WHERE ROWID in (SELECT row_id
      4  FROM exceptions)
      5  FOR UPDATE;
```

ROWID	ID	LAST_NAME	DEPT_ID
AAAAeyAADAAAAA1AAA	1003	Pirie	50

1 row selected.

4 Correct the errors in data:

```
SQL> UPDATE summit.employee
      2  SET id=10
      3  WHERE rowid='AAAAeyAADAAAAA1AAA';
1 row processed.
SQL> COMMIT;
Statement processed.
```


How to Identify Row Violation (continued)

- 5 Truncate the EXCEPTIONS table and reenable the constraint:

```
SQL> TRUNCATE TABLE exceptions;
```

Statement processed.

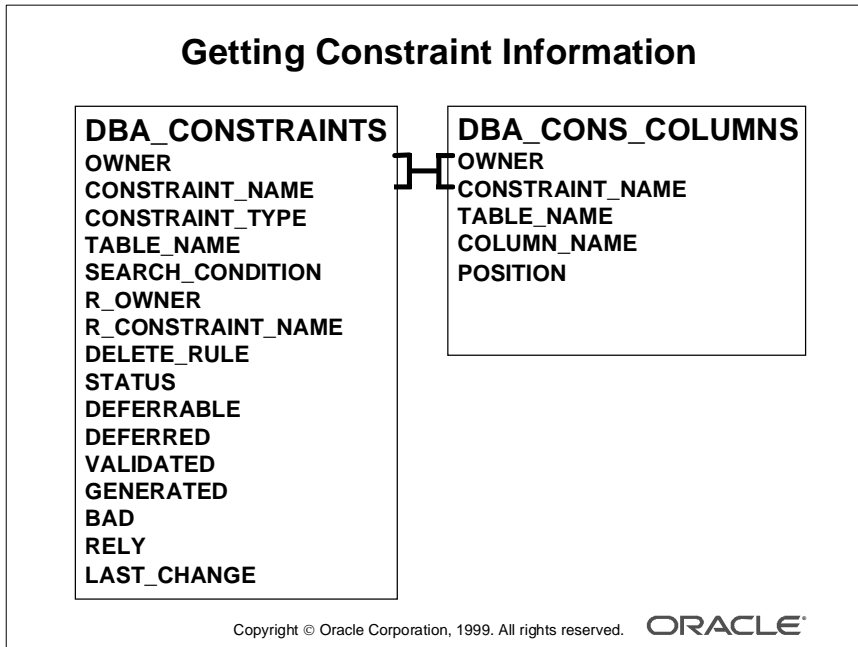
```
SQL> ALTER TABLE summit.employee
```

```
2  ENABLE VALIDATE CONSTRAINT employee_dept_id_fk
```

```
3  EXCEPTIONS INTO system.exceptions;
```

Statement processed.

Getting Constraint Information



Constraints and Their Status

Use the following query to obtain the names, types, and status of all constraints on SUMMIT's EMPLOYEE table:

```
SQL> SELECT constraint_name, constraint_type, deferrable,
2  deferred, validated
3  FROM dba_constraints
4  WHERE owner='SUMMIT'
5  AND table_name='EMPLOYEE' ;
```

CONSTRAINT_NAME	C	DEFERRABLE	DEFERRED	VALIDATED
-----	-	-----	-----	-----
EMPLOYEE_DEPT..	R	DEFERRABLE	DEFERRED	VALIDATED
EMPLOYEE_ID_PK	P	DEFERRABLE	IMMEDIATE	VALIDATED
SYS_C00565	C	NOT DEFERRABLE	IMMEDIATE	VALIDATED

3 rows selected.

Constraints and Their Status (continued)

The following table shows the columns in the DBA_CONSTRAINTS view that are not self-evident.

Name	Description
CONSTRAINT_TYPE	The type of constraint is P if Primary key, U if Unique, R if Foreign key, or C if Check constraint. NOT NULL constraints are stored as check constraints.
SEARCH_CONDITION	Shows the condition specified for a check constraint
R_OWNER R_CONSTRAINT_NAME	Defines the owner and name of the referenced constraint for foreign keys
GENERATED	Indicates whether the constraint name is system-generated (Valid values are 'USER NAME' and 'GENERATED NAME'.)
BAD	Indicates that the constraint is to be rewritten to avoid such situations as Year 2000 problems (This might happen because earlier releases of Oracle allowed 2-digit years to be specified in check constraints.)
RELY	If set, this flag will be used in the optimizer
LAST_CHANGE	The date when the constraint was last enabled or disabled

Columns in Constraints

To get the columns in the constraints on SUMMIT's EMPLOYEE table, use the following query:

```
SQL> SELECT c.constraint_name, c.constraint_type,
2  cc.column_name
3  FROM dba_constraints c, dba_cons_columns cc
4  WHERE c.owner='SUMMIT'
5  AND c.table_name='EMPLOYEE'
6  AND c.owner = cc.owner
7  AND c.constraint_name = cc.constraint_name
8  ORDER BY cc.position;
```

Columns in Constraints (continued)

CONSTRAINT_NAME	C	COLUMN_NAME
-----	-	-----
EMPLOYEE_DEPT...	R	DEPT_ID
EMPLOYEE_ID_PK	P	ID
SYS_C00565	C	LAST_NAME

3 rows selected.

Finding Primary Key–Foreign Key Relationships

To find foreign keys on SUMMIT's EMPLOYEE table and the parent constraints, use the following query:

```
SQL> SELECT c.constraint_name AS "Foreign Key",
2 p.constraint_name AS "Referenced Key",
3 p.constraint_type,
4 p.owner,
5 p.table_name
6 FROM dba_constraints c, dba_constraints p
7 WHERE c.owner='SUMMIT'
8 AND c.table_name='EMPLOYEE'
9 AND c.constraint_type='R'
10 AND c.r_owner=p.owner
11 AND c.r_constraint_name = p.constraint_name;
```

Foreign Key	Referenced Key	C	OWNER	TABLE_NAME
-----	-----	-	-----	-----
EMPLOYEE_DEPT..	DEPT_PK	P	SUMMIT	DEPARTMENT

1 row selected.

Summary

Summary

In this lesson, you should have learned how to:

- **Implement data integrity**
- **Use an appropriate strategy for creating and maintaining constraints**
- **Obtain information from the data dictionary**

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

Quick Reference

Context	Reference
Initialization parameters	None
Dynamic performance views	None
Data dictionary views	DBA_CONSTRAINTS DBA_CONS_COLUMNS
Commands	CREATE TABLE ... CONSTRAINT ALTER TABLE ADD CONSTRAINT ... EXCEPTIONS INTO ALTER TABLE ... DISABLE CONSTRAINT ALTER TABLE ... ENABLE NOVALIDATE CONSTRAINT ALTER TABLE ... ENABLE VALIDATE CONSTRAINT ... EXCEPTIONS INTO
Packaged procedures and functions	None

Loading Data

Objectives

Objectives

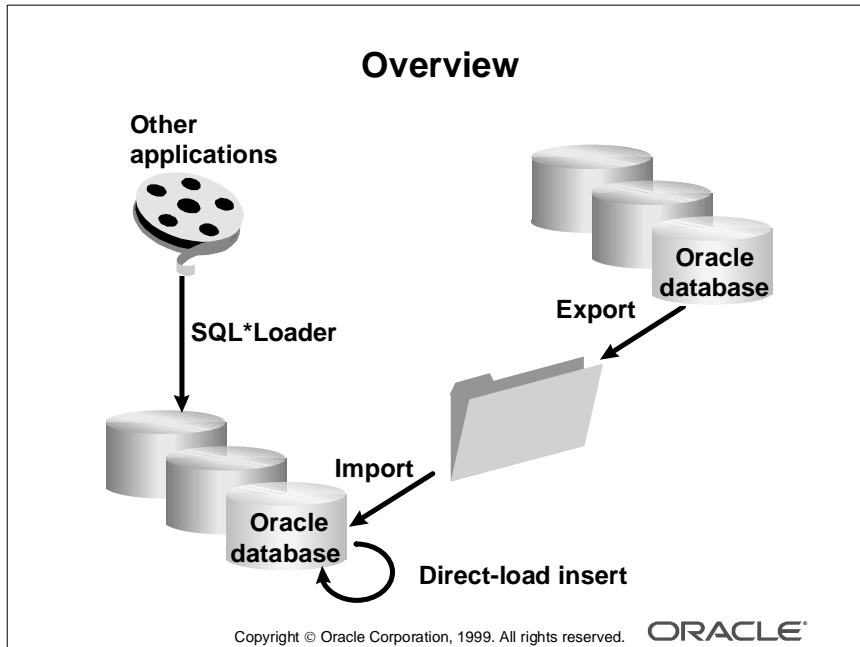
After completing this lesson, you should be able to do the following:

- Load data using direct-load insert
- Load data into Oracle tables using SQL *Loader:
 - Conventional path
 - Direct path

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Overview



Loading Data

Several methods are available for loading data into tables in an Oracle database. The methods that will be discussed here are:

- Direct-load insert
- SQL*Loader
- Export and Import utilities

Direct-Load Insert

Direct-load inserts can be used to copy data from one table to another table within the same database. It speeds up the insert operation, bypassing the buffer cache and writing data directly into the data files.

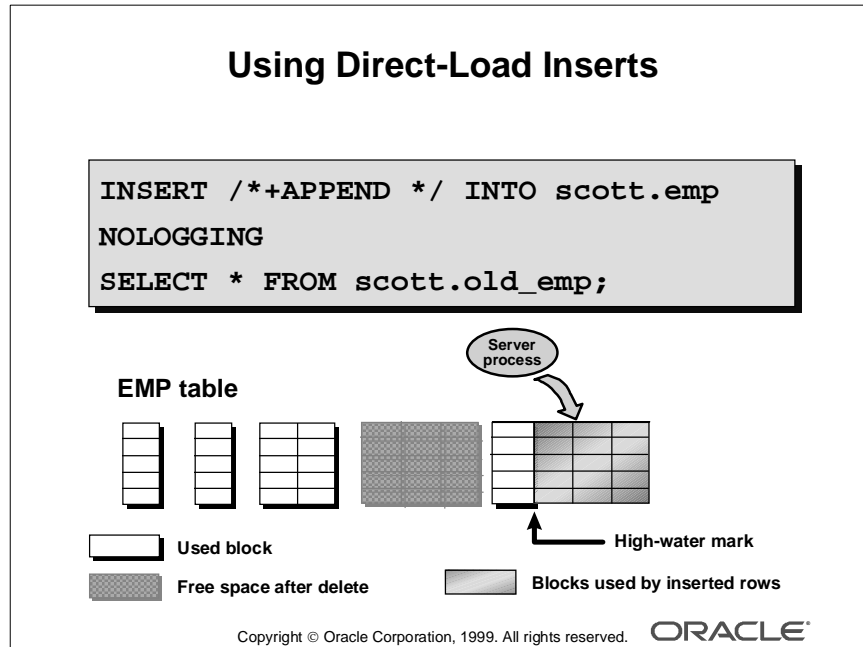
SQL*Loader

SQL*Loader is a utility that is used to load data from external files into Oracle tables. It provides a means of migration from other systems to the Oracle database.

Export and Import Utilities

The Export utility enables users to extract dictionary information and data from an Oracle database and move them into an operating system file in Oracle-binary format. The files generated by Export can be read by the Import utility into the same Oracle database or a different Oracle database. The Export and Import utilities will be discussed in the next lesson.

Loading Data Using Direct-Load Insert



Syntax

A direct-load insert can be invoked by using the APPEND hint, as shown in the command below:

```

INSERT /*+APPEND */ INTO [ schema. ] table
[ [NO]LOGGING ]
sub-query;
  
```

where:	schema table sub-query	is the owner of the table is the name of the table is the subquery used to select the required columns and rows for insert
--------	------------------------------	---

Direct-load inserts are only available when the INSERT INTO SELECT command is used. This option is not available when INSERT INTO VALUES command is used. The direct-load insert can be used both on nonpartitioned and partitioned tables. This operation maintains indexes and also enforces all enabled constraints. It also enables other users to modify other rows in the table concurrently.

LOGGING Mode

When inserting using the LOGGING option, which is the default, the operation generates redo log entries, making complete recovery possible in case of failures.

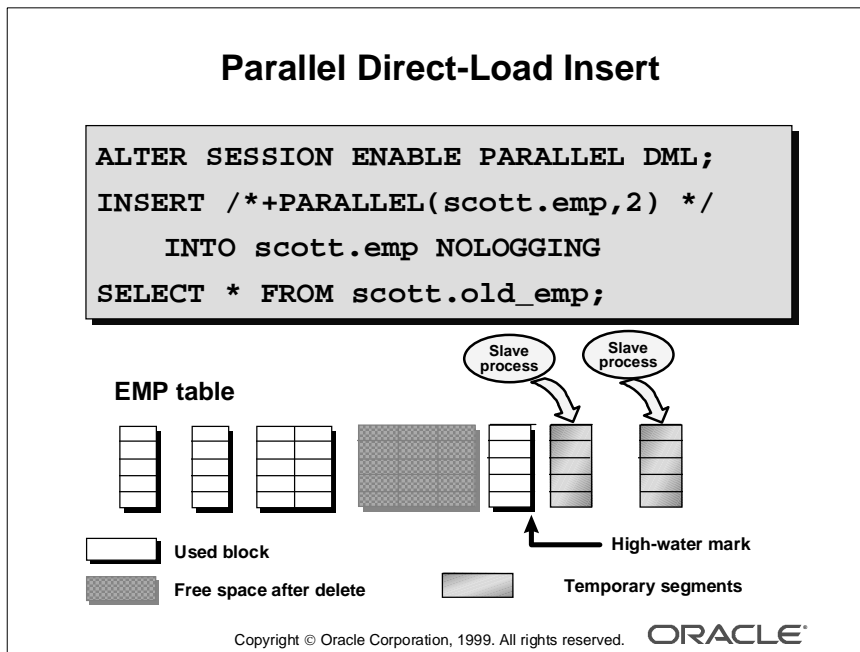
If the NOLOGGING option is used, changes to data are not recorded in the redo log buffer. Some minimal logging still occurs for operations that update the data dictionary. The NOLOGGING mode is used if this attribute has been set for the table.

If several online modifications to the data in the table are likely to occur subsequently, it might be advisable to set the NOLOGGING attribute before the load and reset it to LOGGING once the load is completed.

Other Considerations

Direct-load inserts allow other transactions to make changes to the table concurrently.

All data inserted using this method is loaded above the high-water mark. If the table contains many blocks where rows have been deleted, space may be wasted, and full table scans may be slower.



Parallel Direct-Load Inserts

Direct-load inserts can be made in parallel using one of the following methods:

- Using a PARALLEL hint in the INSERT statement, as in the example
- Creating the table or altering it to specify the PARALLEL clause

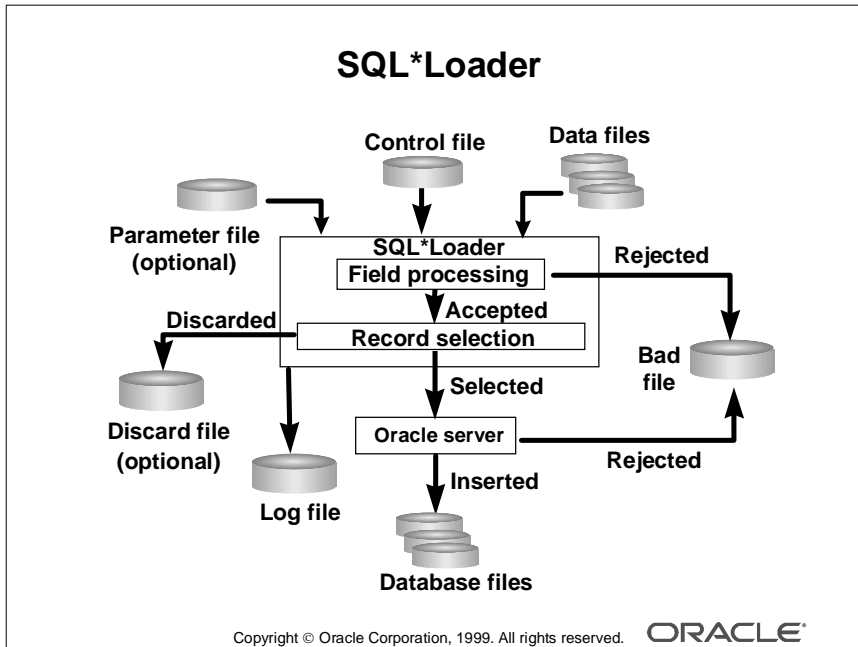
When parallel direct-load inserts are made, the Oracle server uses several processes, known as parallel query slaves, to insert data into the table. Temporary segments are allocated to store the data inserted by each slave process. When the transaction commits, the extents in these individual segments become a part of the table in which records are inserted.

Note

- The ALTER SESSION ENABLE PARALLEL DML command must be executed at the beginning of a transaction.
- An object that is modified using parallel direct-load insert cannot be queried or modified again within the same transaction.

For a detailed discussion of parallel direct-load inserts, see the *Oracle8i Reference*, “Parallel Execution.”

Loading Data Using SQL*Loader



SQL*Loader Features

SQL*Loader loads data from external files into tables in an Oracle database.

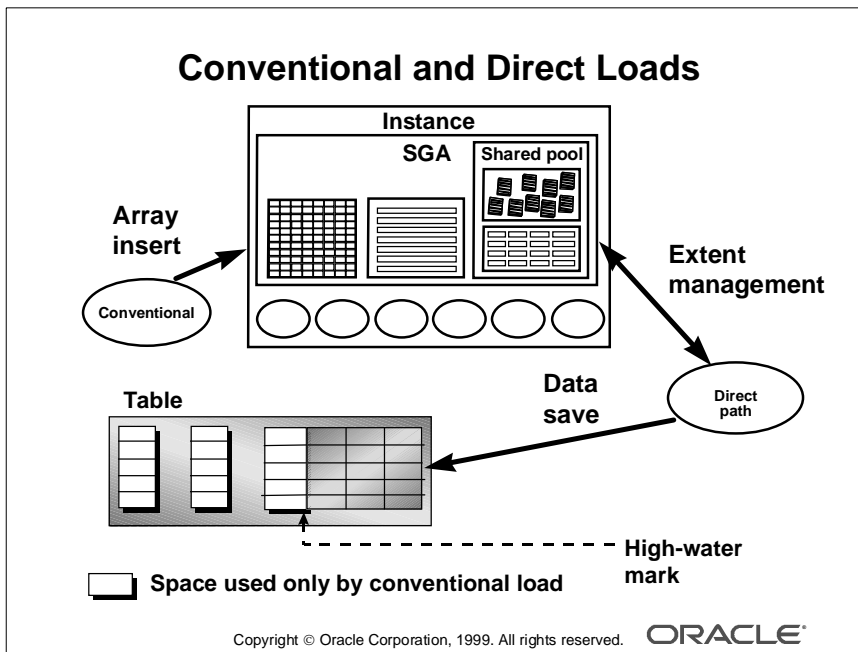
SQL*Loader has the following features:

- One or more input files can be used.
- Several input records can be combined into one logical record for loading.
- Input fields can be fixed or variable length.
- Input data can be in any format—character, binary, packed decimal, date, and zoned decimal.
- Data can be loaded from different types of media such as disk, tape, or named pipes.
- Data can be loaded into several tables in one run.
- Options are available to replace or to append to existing data in the tables.
- SQL functions can be applied on the input data before the row is stored in the database.
- Column values can be autogenerated based on rules. For example, a sequential key value can be generated and stored in a column.
- Data can be loaded directly into the table, bypassing the database buffer cache.

Files Used by SQL*Loader

SQL*Loader uses the following files:

- Control file: Specifies the input format, output tables, and optional conditions that may be used to load only part of the records found in the input data files
- Data files: Contain the data in the format defined in the control file
- Parameter file: Is an optional file that can be used to define the command line parameters for the load
- Log file: Is created by SQL*Loader and contains a record of the load
- Bad file: Is used by the utility to write the records that are rejected during the load (This can occur during input record validation by the utility or during record insertion by the Oracle server.)
- Discard file: Is a file that can be created, if necessary, to store all records that did not satisfy the selection criteria



Loading Methods

SQL*Loader provides two methods for loading data:

- Conventional path
- Direct path

Conventional Path Load

Conventional path load builds an array of rows to be inserted and uses the SQL INSERT statement to load the data. During conventional path loads, input records are parsed based on field specifications, and an array of records is built and inserted into the table specified in the control file. Records that do not conform to the field specifications are rejected and those records that do not satisfy the selection criteria are discarded.

Conventional path loads can be used to load data into both clustered and unclustered tables. Redo log generation is controlled by the logging attribute for the table being loaded.

Direct Path Load

A direct path load builds blocks of data in memory and saves these blocks directly into the extents allocated for the table being loaded. Redo log entries are not generated unless the database is in archivelog mode. Direct path loads use the field specifications to build whole Oracle blocks of data, and write the blocks directly to Oracle data files. Direct path load bypasses the database buffer cache and accesses the SGA only for extent management and adjustment of the high-water mark.

Direct Path Load (continued)

Direct path load is generally faster than conventional path load, but cannot be used in all situations. The next section compares conventional path to direct path loading, and gives examples of situations in which each of them can be used.

Note: The `catldr.sql` script, supplied by Oracle, creates views that are used by direct path load. It is automatically invoked when the `catalog.sql` script is run.

Comparing Direct and Conventional Path Loads

Conventional Load	Direct Path Load
Uses COMMITs to make changes permanent	Uses data saves
Redo log entries always generated	Generates redo only under specific conditions
Enforces all constraints	Enforces only primary key, unique, and NOT NULL
INSERT triggers fire	INSERT triggers do not fire
Can load into clustered tables	Cannot load into clustered tables
Other users can make changes to tables	Other users cannot make changes to tables

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

Method of Saving Data

Conventional path loads use SQL processing and database COMMITs for saving data. The insertion of an array of records is followed by a commit operation. Each data load might involve several transactions.

Direct path loads use *data saves* to write blocks of data to Oracle data files. The following features differentiate a data save from a COMMIT:

- During a data save, only full database blocks are written to the database.
- The blocks are written after the high-water mark of the table.
- After a data save, the high-water mark is moved.
- Internal resources are not released after a data save.
- A data save does not end the transaction.
- Indexes are not updated at each data save.

Logging of Changes

Conventional path loading generates redo log entries just as any DML statement. When using a direct path, load redo log entries are not generated if:

- The database is in NOARCHIVELOG mode
- The database is in ARCHIVELOG mode, but logging is disabled. Logging can be disabled by setting the NOLOGGING attribute for the table or by using the UNRECOVERABLE clause in the control file.

Enforcement of Constraints

During a conventional path load, all constraints that are enabled are enforced as they would be during any DML operation.

During direct loads, the constraints are handled as follows:

- NOT NULL constraints are checked when arrays are built.
- Foreign key and CHECK constraints are disabled, and can be enabled at the end of the run by using the appropriate commands in the control file. Foreign key constraints are disabled because they reference other rows or tables, and CHECK constraints are disabled because they may use SQL functions. If only a small number of rows are to be inserted into a large table, use conventional loads.
- Primary key and unique constraints are checked during and at the end of the run, and may be disabled if they are violated.

Firing of INSERT Triggers

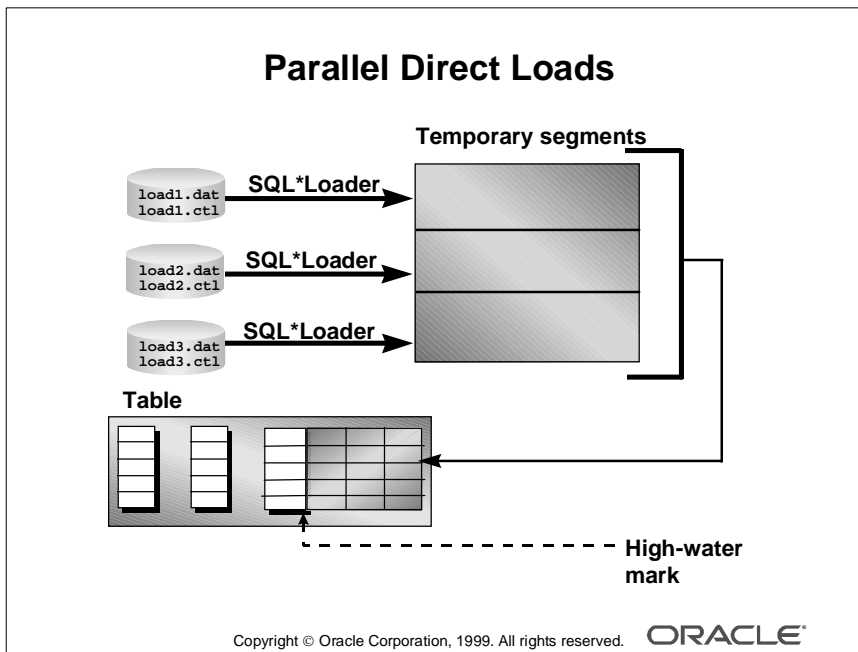
While INSERT triggers are fired during conventional path loads, they are disabled before a direct path load and reenabled at the end of the run. They may remain disabled in the event that a referenced object is not accessible at the end of the run. Consider using conventional path loads to load data into tables with INSERT triggers.

Loading into Clustered Tables

Direct loads cannot be used to load rows into clustered tables. Clustered tables can be loaded using conventional path loads only.

Locking

While a direct load is in progress, other transactions cannot make changes to the tables being loaded. The only exception to this rule is when several parallel direct load sessions are used concurrently. Parallel direct loads are discussed in the following section.



Parallel Direct Loads

Parallel direct loads permit the use of several concurrent direct load sessions to load data into a single table.

Sequence of Operations

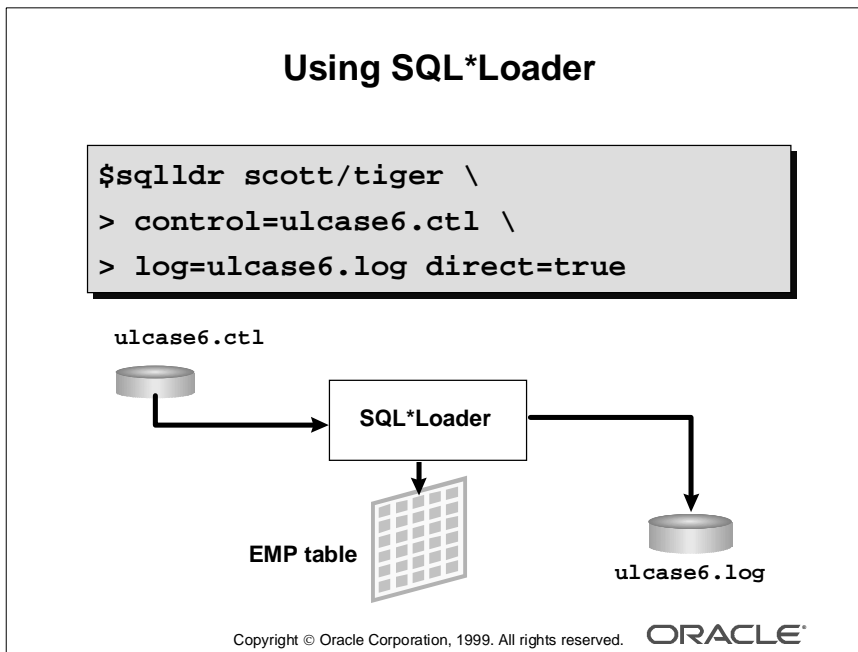
Use different input data files for each parallel direct load session. When several parallel direct load sessions are initiated, the loading is carried out using the following steps:

- 1 Each session uses a temporary segment to load the data from input data files. These temporary segments are created in the tablespace that contains the table being loaded. If the tablespace contains several data files, then for each session, a user can specify the data file where the temporary segment is to be created. The storage parameters for these segments can also be specified by the user. By default, these segments use the same storage attributes as the table being loaded.
- 2 The last extent in each temporary segment is trimmed to deallocate unused space when the session is completed.
- 3 All the temporary segments are combined to form one segment at the end of the run.
- 4 The resulting temporary segment is added to the table segment.

Restrictions

The following restrictions are enforced on parallel direct path loads:

- Indexes are not maintained by the load. Drop indexes before a parallel load and re-create them at the end of the run.
- Referential integrity, check constraints, and triggers must be disabled and have to be reenabled manually.
- Rows can only be appended to existing data, because individual loads are not coordinated. If data in the table needs to be replaced, manually truncate the table before the parallel load.



Command Line

Use the following command on UNIX or Windows NT to perform the data load:

```
$sqlldr [keyword=]value [ [ [,] keyword=]value ] ...
```

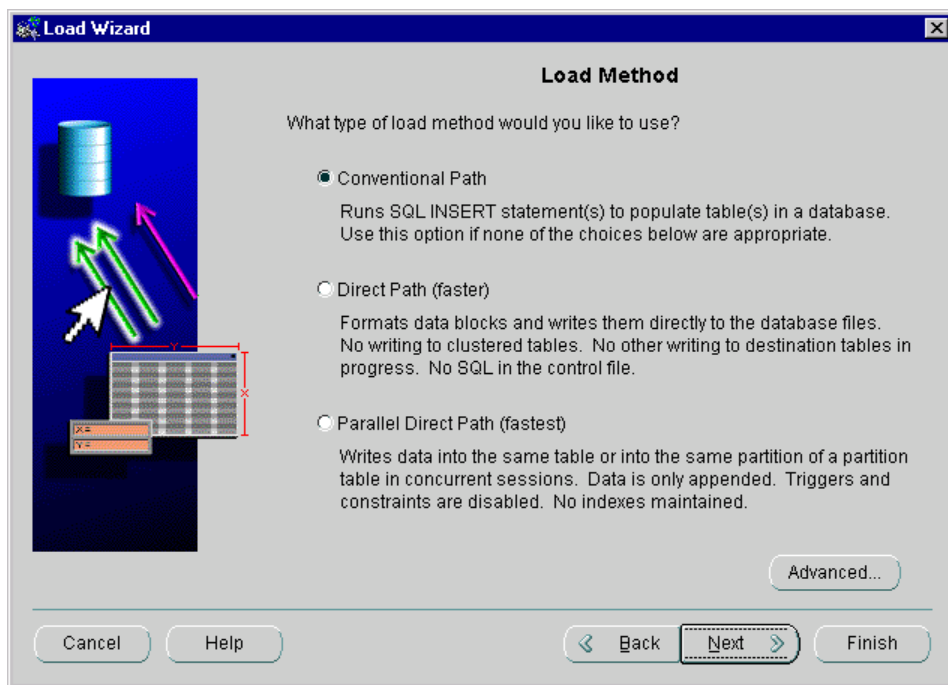
where: keyword Is one of the keywords discussed in the next section
 value Is the value assigned to the keyword

Note

- If keywords are not specified, the values must be specified in the correct order. Although this option is available, it is generally advisable to use the keywords.
- As shown in the slide, it is possible to specify the first few values without keywords and then specify other values with keywords.

How to Use Oracle Enterprise Manager to Load Data

- 1 Launch the Oracle Enterprise Manager console:
Start—>Programs—>Oracle - *EMV2 Home*—>Oracle Enterprise Management
—>Enterprise Manager Console
- 2 Enter Administrator, Password, and Management Server. Click OK to log in to the console.
- 3 Expand the Databases folder.
- 4 Select your working database and choose Data Management—>Load from the right mouse menu.
- 5 Enter the control filename in the Control File page, and click Next.
- 6 Enter names of other files in the Data Files page, and click Next.
- 7 Specify load options in the Load Method page, and click Next.



- 8 Specify a schedule in the Scheduling page and click Finish.
- 9 Verify options on the Summary page and click OK.

Command Line Keywords

Some of the commonly used keywords and load options to be specified on page 3 of the Data Manager Wizard are shown below.

Keyword	Meaning
USERID	Oracle username and password. If password is not specified, the user will be prompted for the password.
CONTROL	Control filename
LOG	Filename of work log; defaults to control filename with an extension of log
BAD	Filename of file that stores all rejected records; defaults to the control filename with an extension of bad
DATA	Input data filenames
DISCARD	Filename for optional discard file where records not selected are stored
DISCARDMAX	Maximum number of discards to allow; the default is to allow all discards. Use this parameter as a safety measure to stop the run if the wrong input files are specified.
SKIP	The number of records to skip; primarily used to continue a load that failed. Use this option only if loading to a single table or to skip an identical number of records for all the tables loaded.
LOAD	Specifies the number of records to load, after skipping the records specified by SKIP
ERRORS	Maximum number of bad records to allow
ROWS	Specifies the number of rows in the array to be built before each insert for conventional loads. For direct path loads, this defines the approximate number of rows read from input for each data save. Direct load builds full blocks, and then rejects discards and invalid rows before a data save.
BINDSIZE	Specifies the maximum number of bytes to be used for building an array of rows to be inserted in each database call, for conventional loads (If the ROWS parameter is also specified, SQL*Loader will build as many rows as defined by ROWS, subject to the limit imposed by BINDSIZE.)
DIRECT	SQL*Loader uses the direct path if this parameter is set to TRUE. Conventional path, which is the default, is used otherwise.

Keyword	Meaning
PARFILE	Specifies the name of the file that contains all the load parameters (Parameters defined in the command line will override any values defined in the parameter file.)
PARALLEL	This parameter, which is only valid for direct loads, specifies that multiple parallel direct loads are to be carried out.
FILE	Specifies the file in which the temporary segment for a parallel direct load is to be created

The parameters can also be defined within the control file.

Note: The parameters defined here are not exhaustive. For a complete reference, see *Oracle8i Utilities*, “SQL*Loader Command-Line Reference.”

SQL*Loader: Input Files

- **Parameter file contains load options**
- **Data file contains input records**
- **Control file contains load instructions**

```
LOAD DATA
INFILE 'ulcase6.dat'
INSERT
  INTO TABLE emp
  (empno POSITION(01:04) INTEGER
   EXTERNAL NULLIF empno=BLANKS,
   ...
  )
```

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE

Parameter File

The parameter file can be used to specify the parameters for a load. Use a parameter file to store all the commonly used parameters. This file uses the following format to define the parameters:

KEYWORD=VALUE

Control File

The contents of a control file include the following:

- Names of the input data files, using the INFILE clause
- The composition of a logical record from a physical record in the input data files, using clauses such as CONCATENATE and CONTINUEIF
- Field specifications, including position, data types, and delimiters condition specifications, using the FIELDS clause
- Names of tables, using the INTO TABLE clause
- Method of loading; whether data is to be loaded into an empty table, inserted after deleting or truncating existing records, or appended to the existing data
- Records to be skipped for each table, using the CONTINUE_LOAD clause
- Conditions to be used for selecting records to be loaded, using the WHEN clause
- Columns to be loaded

Control File (continued)

- Rules for generating column values, using clauses such as RECNUM and SYSDATE and applying SQL functions
- Column operations, such as trimming and substituting zeros with nulls
- Load parameters, using the OPTIONS clause
- Storage specifications for temporary segments created during a parallel direct load
- Comments prefixed by "--"
- Other direct load options, such as:
 - SINGLEROW to maintain indexes on a row-by-row basis
 - REENABLE to enable disabled constraints at the end of the run
 - SORTED_INDEXES to specify if data is presorted
 - UNRECOVERABLE to suppress generation of redo

Note

- Setting the NOLOGGING attribute for the table using the NOLOGGING keyword is equivalent to using the UNRECOVERABLE option in the control file.
- Data can also be stored inside the control file by specifying INFILE *, and using BEGINDATA to mark the beginning of the data. If this option is used, all records following BEGINDATA will be interpreted as data.
- Several case studies illustrating different load options and control file items can be found in the \$ORACLE_HOME/rdbms/demo directory in UNIX and in the %ORACLE_HOME%\RDBMS80\LOADER directory in Windows NT. The "SQL*Loader Case Studies" chapter in the manual, *Oracle8i Utilities*, explains these cases in detail.
- Complete details of control file commands can be found in the "SQL*Loader Control File Reference" chapter in the manual, *Oracle8i Utilities*.

Data File

The data file contains the records to be processed in the format defined in the control file.

Log File Contents

- **Header information**
- **Global information: Parameters and file names**
- **Table information: Table and column specifications**
- **Data file information: Records processed**
- **Table load information: Errors and discards**
- **Summary statistics**

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

Log Files

The log file is mandatory and the load terminates if the log file cannot be created because of lack of space or permission. The log file contains:

- Header information such as the date of the run and software version number
- Global information including:
 - Names of all input and output files
 - Command line arguments
- Table information such as:
 - Table names
 - Load conditions and method
- Field and column information
- Data file information showing records rejected and records discarded with reasons

Log Files (continued)

- Table load information such as:
 - Number of rows loaded
 - Number of rows that qualified for loading but were rejected due to data errors
 - Number of rows that were discarded
 - Number of rows whose relevant fields were all null
- Summary statistics that include the following data:
 - Amount of space allocated for the array
 - Load statistics for all data files
 - Beginning and ending time of run
 - Total elapsed time
 - Total CPU time including time for all file I/O but excluding any CPU time used by background processes

SQL*Loader: Other Output Files

- **Bad file**
 - Rejected records
 - Same format as data files
- **Discard file**
 - Records not satisfying conditions
 - Same format as data files

Copyright © Oracle Corporation, 1999. All rights reserved. **ORACLE**

Bad File

The bad file contains records that are rejected during processing due to one of the following reasons:

- Input records have errors such as insufficient number or badly formatted fields
- Rows could not be inserted for reasons such as constraint violation

The records in the bad file are in the same format as the input records. The records can be used, after rectifying the errors, to reload the data.

Discard File

The discard file contains data in the same format as the input data files and is useful if data needs to be selectively loaded into tables, either in different databases or at different points in time.

SQL*Loader: Usage Guidelines

- **Use a parameter file to specify commonly used command line options**
- **Place data within the control file only for a small, one-time load**
- **Improve performance by:**
 - **Allocating sufficient space**
 - **Sorting the data on the largest index**
 - **Specifying different files for temporary segments for parallel loads**

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE®

SQL*Loader Usage Guidelines

Use the following guidelines when using SQL*Loader to minimize errors and improve performance:

- Use a parameter file to specify commonly used command line options. For example, if loading into a data warehouse every week, all options except the names of the files may be the same.
- Separating the control file and the data file permits reusing control files for several load sessions.
- Preallocating space based on the expected data volume prevents dynamic allocation of extents during the load and improves the speed of the load.
- When direct loads are used, temporary segments are used to generate indexes for the new data. These indexes are merged with the existing indexes at the end of the load. By sorting the input data on the keys in the largest index, use of sort space can be minimized.
- With parallel direct loads, you can specify the location of temporary segments used for inserting data. For each load session, specify a different database file to achieve maximum performance.

SQL*Loader: Troubleshooting

- **Insufficient space for table or index**
- **Instance failure during the load**
- **If the SORTED INDEXES clause is used and data is not in the order specified**
- **Duplicate keys found in a unique index, unique or primary key during a direct load**
- **BINDSIZE cannot contain a single row**
- **Errors exceed specified limit**
- **Discards exceed specified limit**

Copyright © Oracle Corporation, 1999. All rights reserved. ORACLE®

Troubleshooting SQL*Loader

When a load terminates abnormally, all data that has been loaded up to the point of failure is likely to have been committed. After rectifying the problem, proceed as follows to complete the load:

- If loading to one table or if all tables have the same number of records processed, use the SKIP command line parameter to continue the load.
- If many tables were loaded and the number of records processed is not the same for all tables, use the CONTINUE_LOAD option in the control file to specify the number records to skip for each table.
- Check for any indexes marked unusable. This occurs if the index is not consistent with the table. Drop the indexes in this state and re-create them after completing the load.

Problem Resolution

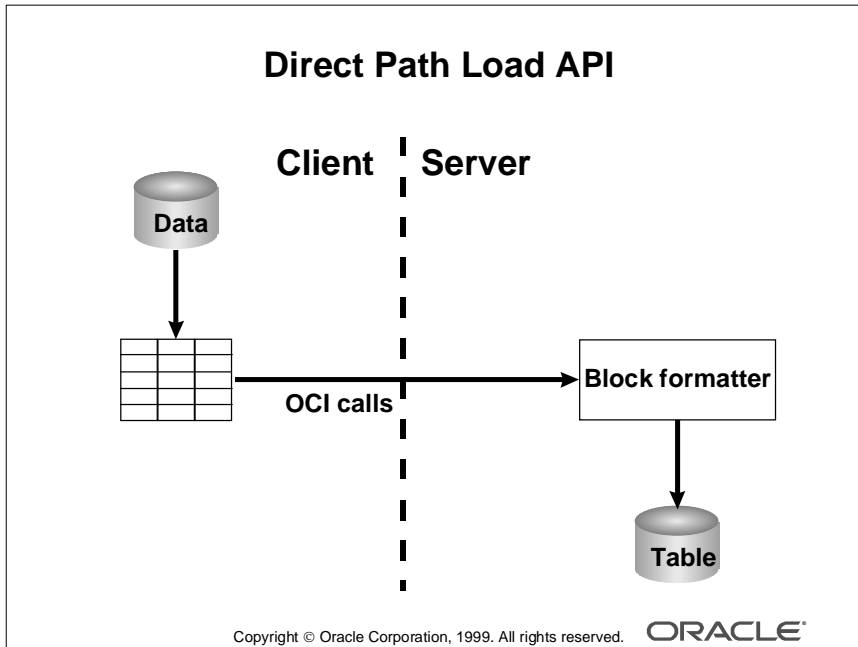
- **Insufficient space:** A load may terminate if the Oracle server could not allocate sufficient space to the tables being loaded. In this case, investigate the cause of the problem—whether it is due to insufficient disk space, or the files becoming full, or MAXEXTENTS being reached, and correct the problem.
- **Instance failure:** Investigate the reason for failure, rectify it, restart the instance, and continue the load.

Problem Resolution (continued)

- Data is not in the order specified: The data is loaded, but the index is in an unusable state. Drop and re-create the indexes that are unusable.
- Duplicate values in a primary key or unique column or unique index: This does not abort the load process, but may result in disabled constraints or unusable indexes. In the case of constraint errors, use an exceptions table to trap the errors and rectify them. If a unique index is unusable, you may have to detect errors by attempting to create a unique constraint on the indexed column, trapping the errors into an exception table, and correcting them.
- BINDSIZE too small: Use a larger value and load the data.
- Errors exceeding the limit set: This occurs when a load is aborted because the number of invalid records has exceeded the ERRORS value specified. The most common cause of this problem is the use of incorrect input data files. Check and use the correct files.
- Discards exceeding the limit set: This occurs when a load is aborted because the number of discarded records has exceeded the DISCARDMAX value specified. The most common cause of this problem is the use of incorrect input data files. Check and use the correct files.

Note: The SORTED INDEXES clause is only applicable to direct path loads. For syntax and details of usage, refer to the “SQL*Loader Control File Reference” chapter in the manual, *Oracle8i Utilities*.

Direct Path Loading



Direct Path Load with Oracle Call Interface

The direct path load interface allows an Oracle Call Interface (OCI) application to access the direct path load engine of the Oracle database server to perform the functions of the Oracle SQL*Loader utility. This functionality provides the ability to load data from external files into an Oracle table.

The OCI direct path load interface has the ability to load multiple rows by loading a direct path stream that contains data for multiple rows.

A direct load operation requires that the object being loaded is locked to prevent DML on the object.

Note that queries are lock free and are allowed while the object is being loaded.

The direct path load interface has the same limitations as SQL*Loader.

Summary

Summary

In this lesson, you should have learned how to:

- Use direct-load insert to copy tables
- Use SQL*Loader to load data
- Use direct path mode when applicable

Copyright © Oracle Corporation, 1999. All rights reserved.

ORACLE®

Quick Reference

Context	Reference
Initialization parameters	None
Dynamic performance views	None
Data dictionary views	None
Commands	sqlldr or sqlload
Packaged procedures and functions	None

