Oracle9*i* Performance Tuning

Student Guide . Volume 1

D11299GC10
Production 1.0
July 2001
D33513

Authors

Peter Kilpatrick
Shankar Raman
Jim Womack

Technical Contributors and Reviewers

Mirza Ahmad
Harald Van Breederode
Howard Bradley
Howard Ostrow
Alexander Hunold
Joel Goodman
John Watson
Michele Cyran
Pietro Colombo
Ranbir Singh
Ruth Baylis
Sander Rekveld
Tracy Stollberg
Connie Dialeris
Wayne Stokes
Scott Gossett
Sushil Kumar
Benoit Dagerville

Contents

Preface

1 Overview of Oracle9*i* Performance Tuning
Objectives 1-2
Tuning Questions 1-3
Tuning Phases 1-5
Tuning Goals 1-6
Examples of Measurable Tuning Goals 1-7
Common Tuning Problems 1-8
Results of Common Tuning Problems 1-9
Proactive Tuning Considerations During Development 1-10
Tuning Steps During Production 1-11
Performance versus Safety Trade-Offs 1-12
Summary 1-13

2 Diagnostic and Tuning Tools
Objectives 2-2
Maintenance of the Alert Log File 2-3
Tuning Using the Alert Log File 2-4
Background Processes Trace Files 2-5
User Trace Files 2-6
Views, Utilities, and Tools 2-7
Dictionary and Special Views 2-9
Dynamic Troubleshooting and Performance Views 2-10
Topics for Troubleshooting and Tuning 2-11
Collecting Systemwide Statistics 2-13
Collecting Session-Related Statistics 2-16
STATSPACK 2-18
STATSPACK Output 2-20
UTLBSTAT and UTLESTAT Utilities 2-23
Oracle Wait Events 2-24
The V$EVENT_NAME View 2-25
Statistics Event Views 2-26
The V$SYSTEM_EVENT View 2-27
The V$SESSION_WAIT View 2-28
Performance Manager 2-31
Overview of Oracle Expert Tuning Methodology 2-33
DBA-Developed Tools 2-34
Summary 2-35

3 Sizing the Shared Pool
Objectives 3-2
The System Global Area 3-3
The Shared Pool 3-4
The Library Cache 3-5
Tuning the Library Cache 3-7
Terminology 3-9
Diagnostic Tools for Tuning the Library Cache 3-10
Are Cursors Being Shared? 3-11
Guidelines: Library Cache Reloads 3-12
Invalidations 3-14
Sizing the Library Cache 3-16
Cached Execution Plans 3-17
View to Support Cached Execution Plans 3-18
Global Space Allocation 3-20
Large Memory Requirements 3-22
Tuning the Shared Pool Reserved Space 3-24
Keeping Large Objects 3-26
Anonymous PL/SQL Blocks 3-28
Other Parameters Affecting the Library Cache 3-30
The Data Dictionary Cache, Terminology, and Tuning 3-32
Diagnostic Tools for Tuning the Data Dictionary Cache 3-33
Measuring the Dictionary Cache Statistics 3-34
Tuning the Data Dictionary Cache 3-36
Guidelines: Dictionary Cache Misses 3-37
UGA and Oracle Shared Server 3-38
Sizing the User Global Area 3-39
Large Pool 3-40
Summary 3-41

4 Sizing the Buffer Cache
Objectives 4-2
Overview 4-3
Buffer Cache Sizing Parameters in Oracle9*i* 4-5
Dynamic SGA Feature in Oracle9*i* 4-6
Unit of Allocation in the Dynamic SGA 4-7
Granule 4-8
Allocating Granules at Startup 4-9
Adding Granules to Components 4-10
Dynamic Buffer Cache Size Parameters 4-11

Example: Increasing the Size of an SGA Component 4-12
Deprecated Buffer Cache Parameters 4-13
Dynamic Buffer Cache Advisory Parameter 4-14
View to Support Buffer Cache Advisory 4-15
Using V$DB_CACHE_ADVICE 4-16
Managing the Database Buffer Cache 4-17
Tuning Goals and Techniques 4-20
Diagnostic Tools 4-22
Measuring the Cache Hit Ratio 4-23
Guidelines for Using the Cache Hit Ratio 4-24
Guidelines to Increase the Cache Size 4-25
Using Multiple Buffer Pools 4-27
Defining Multiple Buffer Pools 4-28
Enabling Multiple Buffer Pools 4-30
KEEP Buffer Pool Guidelines 4-31
RECYCLE Buffer Pool Guidelines 4-32
Calculating the Hit Ratio for Multiple Pools 4-35
Identifying Candidate Pool Segments 4-36
Dictionary Views with Buffer Pools 4-37
Caching Tables 4-38
Other Cache Performance Indicators 4-39
Free Lists 4-41
Diagnosing Free List Contention 4-42
Resolving Free List Contention 4-44
Auto-management of Free Space 4-45
Summary 4-46

5 Sizing Other SGA Structures
Objectives 5-2
The Redo Log Buffer 5-3
Sizing the Redo Log Buffer 5-4
Diagnosing Redo Log Buffer Inefficiency 5-5
Using Dynamic Views to Analyze Redo Log Buffer Efficiency 5-6
Redo Log Buffer Tuning Guidelines 5-8
Reducing Redo Operations 5-10
Monitoring Java Pool Memory 5-12
Sizing the SGA for Java 5-13
Multiple I/O Slaves 5-14
Multiple DBWR Processes 5-15
Tuning DBWn I/O 5-16
Summary 5-17

6 **Database Configuration and I/O Issues**
Objectives 6-2
Oracle Processes and Files 6-3
Performance Guidelines 6-4
Distributing Files across Devices 6-5
Tablespace Usage 6-6
Diagnostic Tools for Checking I/O Statistics 6-7
I/O Statistics 6-9
File Striping 6-10
Tuning Full Table Scan Operations 6-12
Checkpoints 6-15
Diagnostic Tools for Tuning Checkpoint Performance 6-16
Guidelines 6-17
Defining and Monitoring FASTSTART Checkpointing 6-19
Redo Log Groups and Members 6-20
Online Redo Log File Configuration 6-21
Archive Log File Configuration 6-23
Diagnostic Tools 6-24
Summary 6-25

7 **Optimizing Sort Operations**
Objectives 7-2
The Sorting Process 7-3
Sort Area and Parameters 7-4
New Sort Area Parameters 7-6
Sort Area and Parameters 7-7
Tuning Sorts 7-9
The Sorting Process and Temporary Space 7-10
Temporary Space Segments 7-11
Operations Requiring Sorts 7-12
Avoiding Sorts 7-14
Diagnostic Tools 7-16
Diagnostics and Guidelines 7-18
Monitoring Temporary Tablespaces 7-19
Temporary Tablespace Configuration 7-20
Summary 7-22

8 Diagnosing Contention for Latches
Objectives 8-2
Latches: Overview 8-3
Purpose of Latches 8-4
Waiting for a Latch 8-5
Latch Request Types 8-6
Latch Contention 8-7
Reducing Contention for Latches 8-9
Important Latches for the DBA 8-10
Shared Pool and Library Cache Latches 8-12
Summary 8-13

9 Tuning Rollback Segments
Objectives 9-2
Rollback Segments: Usage 9-3
Rollback Segment Activity 9-4
Rollback Segment Header Activity 9-5
Growth of Rollback Segments 9-6
Tuning the Manually Managed Rollback Segments 9-7
Diagnostic Tools 9-8
Diagnosing Contention for Manual Rollback Segment Header 9-9
Guidelines: Number of Manual Rollback Segments (RBSs) 9-11
Guidelines: Sizing Manual Rollback Segments 9-13
Sizing Transaction Rollback Data 9-14
Using Less Rollback 9-18
Possible Problems Caused by Small Rollback Segments 9-20
Automatic Undo Management in Oracle9*i* 9-21
Tablespace for Automatic Undo Management 9-22
Altering an Undo Tablespace 9-23
Switching Undo Tablespaces 9-24
Dropping an Undo Tablespace 9-25
Parameters for Automatic Undo Management 9-26
Monitoring Automatic Undo Management 9-28
Summary 9-29

10 Monitoring and Detecting Lock Contention
Objectives 10-2
Locking Mechanism 10-3
Two Types of Locks 10-6
DML Locks 10-8
Table Lock Modes 10-10
DML Locks in Blocks 10-14
DDL Locks 10-15

Possible Causes of Lock Contention 10-17
Diagnostics Tools for Monitoring Locking Activity 10-18
Guidelines for Resolving Contention 10-20
Deadlocks 10-22
Summary 10-25

11 Tuning the Oracle Shared Server
Objectives 11-2
Overview 11-3
Oracle Shared Server Characteristics 11-4
Monitoring Dispatchers 11-5
Monitoring Shared Servers 11-7
Monitoring Process Usage 11-9
Shared Servers and Memory Usage 11-10
Troubleshooting 11-11
Obtaining Dictionary Information 11-12
Summary 11-13

12 Application Tuning
Objectives 12-2
The Role of the Database Administrator 12-3
Data Storage Structures 12-4
Selecting the Physical Structure 12-5
Data Access Methods 12-7
Clusters 12-8
Cluster Types 12-9
Situations Where Clusters Are Useful 12-10
B-Tree Indexes 12-11
Compressed Indexes 12-13
Bitmap Indexes 12-14
Creating and Maintaining Bitmap Indexes 12-16
B-Tree Indexes and Bitmap Indexes 12-17
Reverse Key Index 12-18
Creating Reverse Key Indexes 12-19
Index-Organized Tables (IOTs) 12-20
Index-Organized Tables and Heap Tables 12-21
Creating Index-Organized Tables 12-22
IOT Row Overflow 12-23
IOT Dictionary Views 12-24
Using a Mapping Table 12-25
Maintaining a Mapping Table 12-26
Materialized Views 12-27
Creating Materialized Views 12-28
Materialized Views: Manual Refreshing 12-30
Query Rewrites 12-31

Materialized Views and Query Rewrites: Example   12-33
Enabling and Controlling Query Rewrites   12-35
Disabling Query Rewrites: Example   12-37
OLTP Systems   12-38
OLTP Requirements   12-39
OLTP Application Issues   12-41
Decision Support Systems (Data Warehouses)   12-42
Data Warehouse Requirements   12-43
Data Warehouse Application Issues   12-45
Hybrid Systems   12-46
Summary   12-47

13 Using Oracle Blocks Efficiently
Objectives   13-2
Database Storage Hierarchy   13-3
Allocation of Extents   13-4
Avoiding Dynamic Allocation   13-5
Locally Managed Extents   13-6
Pros and Cons of Large Extents   13-7
The High-Water Mark   13-9
Table Statistics   13-11
The DBMS_SPACE Package   13-12
Recovering Space   13-15
Database Block Size   13-16
The DB_BLOCK_SIZE Parameter   13-17
Small Block Size: Pros and Cons   13-18
Large Block Size: Pros and Cons   13-19
PCTFREE and PCTUSED   13-20
Guidelines for PCTFREE and PCTUSED   13-22
Migration and Chaining   13-23
Detecting Migration and Chaining   13-24
Selecting Migrated Rows   13-25
Eliminating Migrated Rows   13-26
Index Reorganization   13-28
Monitoring Index Space   13-29
Deciding Whether to Rebuild or Coalesce an Index   13-30
Monitoring Index Usage   13-32
Identifying Unused Indexes   13-33
Summary   13-34

14 SQL Statement Tuning
Objectives 14-2
Optimizer Modes 14-3
Setting the Optimizer Mode 14-4
Optimizer Plan Stability 14-6
Plan Equivalence 14-7
Creating Stored Outlines 14-8
Using Stored Outlines 14-9
Editing Stored Outlines 14-11
Maintaining Stored Outlines 14-13
Using Hints in a SQL Statement 14-14
Overview of Diagnostic Tools 14-15
SQL Reports in STATSPACK 14-16
Explain Plan 14-17
Using SQL Trace and TKPROF 14-18
Enabling and Disabling SQL Trace 14-20
Formatting the Trace File with TKPROF 14-21
TKPROF Statistics 14-23
SQL*Plus AUTOTRACE 14-24
Managing Statistics 14-25
Table Statistics 14-27
Index Statistics 14-28
Column Statistics 14-29
Histograms 14-30
Generating Histogram Statistics 14-31
Copying Statistics Between Databases 14-33
Example: Copying Statistics 14-34
Summary 14-37

15 Tuning the Operating System and Using Resource Manager
Objectives 15-2
Operating System Tuning 15-3
System Architectures 15-4
Virtual and Physical Memory 15-5
Paging and Swapping 15-6
Tuning Memory 15-7
Tuning I/O 15-8
Understanding Different I/O System Calls 15-9
CPU Tuning 15-10
Process versus Thread 15-11
Overview of Database Resource Manager 15-12
Database Resource Management Concepts 15-13

Resource Allocation Methods 15-14
The Original Plan: SYSTEM_PLAN 15-16
Using Subplans 15-17
Administering the Database Resource Manager 15-18
Assigning the Resource Manager Privilege 15-20
Creating Database Resource Manager Objects 15-21
Assigning Users to Consumer Groups 15-25
Setting the Resource Plan for an Instance 15-26
Changing a Consumer Group Within a Session 15-27
Changing Consumer Groups for Sessions 15-28
Database Resource Manager Information 15-29
Current Database Resource Manager Settings 15-32
Guidelines 15-33
Summary 15-34

16 Workshop Overview
Objectives 16-2
Approach to Workshop 16-3
Company Information 16-4
Workshop Database Configuration 16-6
Workshop Procedure 16-7
Choosing a Scenario 16-8
Workshop Scenarios 16-9
Collecting Information 16-10
Generating a Workshop Load 16-11
Results 16-12
Summary 16-13

A Practices Solutions

B Tuning Workshop

C Example of STATSPACK Repot

D Redundant Arrays of Inexpensive Disks Technology (RAID)

Preface

Profile

Before You Begin This Course

Before you begin this course, you should have the following qualifications:

- Thorough knowledge of Oracle Database Administration
- Thorough knowledge of SQL and SQL*Plus
- Working experience with Oracle Database Administration.

Prerequisites

- *Oracle9i DBA Fundamentals I* (*D11321GC10*) (inClass)
- *Oracle9i DBA Fundamentals II* (*D11297GC10*) (inClass)
- *Introduction to Oracle9i: SQL* (*40049GC10*) (inClass)

How This Course Is Organized

Oracle9i Performance Tuning is an instructor-led course featuring lecture and hands-on exercises. Online demonstrations and written practice sessions reinforce the concepts and skills introduced. There is a workshop to be run on the last day that will reinforce the topics learnt during the class.

Related Publications

Oracle Publications

Title

Oracle9i Database Concepts

Oracle9i Performance Methods

Oracle9i Performance Guide and Reference

Typographic Conventions

Typographic Conventions in Text

Convention	Element	Example
Bold italic	Glossary term (if there is a glossary)	The ***algorithm*** 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 to the ORD block. 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: `debug.set ('I", 300);` Directory: `bin` (DOS), `$FMHOME` (UNIX) Filename: Locate the `init.ora` file. Password: User `tiger` as your password. Pathname: Open `c:\my_docs\projects` URL: Go to `http://www.oracle.com` User input: Enter `300` Username: Log on as `scott`
Initial cap	Graphics labels (unless the term is a proper noun)	Customer address (*but* Oracle Payables)
Italic	Emphasized words and phrases, titles of books and courses, variables	Do *not* save changes to the database. For further information, see *Oracle7 Server SQL Language Reference Manual.* Enter `user_id@us.oracle.com`, where *user_id* 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 keys one at a time: [Alternate], [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` *`role`*

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

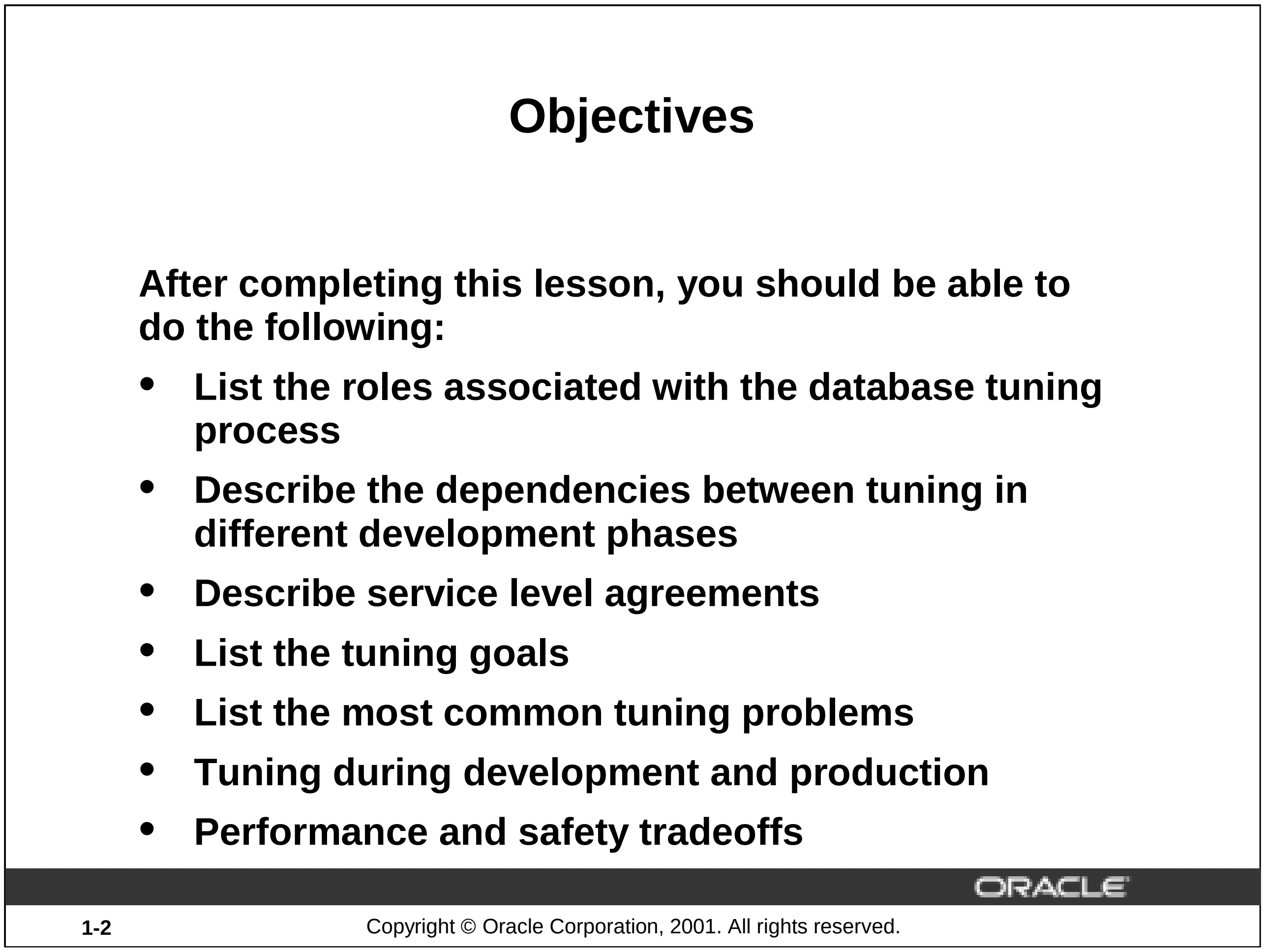
This simplified path translates to the following:

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

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

1

Overview of Oracle 9*i* Performance Tuning

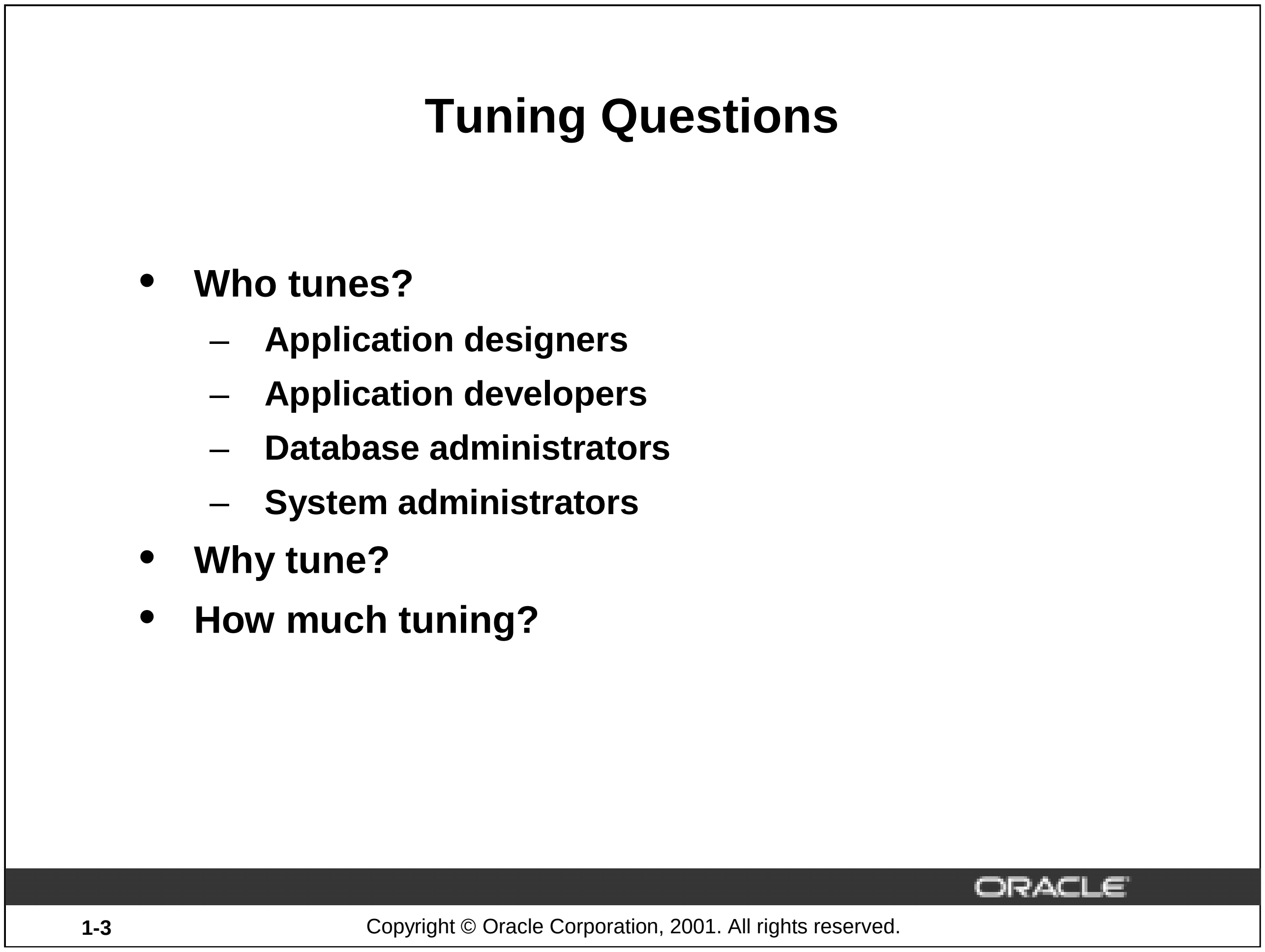


Objectives

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

- List the roles associated with the database tuning process
- Describe the dependencies between tuning in different development phases
- Describe service level agreements
- List the tuning goals
- List the most common tuning problems
- Tuning during development and production
- Performance and safety tradeoffs

1-2



Tuning Questions

- **Who tunes?**
  - **Application designers**
  - **Application developers**
  - **Database administrators**
  - **System administrators**
- **Why tune?**
- **How much tuning?**

ORACLE

1-3

Tuning Questions

Who Tunes?

Everyone involved with the Oracle9*i* system—system architects, designers, developers, and database administrators (DBAs)—should think about performance and tuning in doing their work.

If problems develop, it is usually the DBA who has to make the first attempt at solving them.

Why Tune?

The best practice of tuning is careful design of systems and applications, and the majority of performance gains are realized by tuning the application.

You are much less likely to run into performance problems if:

- The hardware can meet user demands
- Your Oracle9*i* database was carefully designed
- Your application developers write efficient SQL programs

If wrong decisions were made early, or if users expect much more from the system now than they did previously, you should seriously consider further tuning to improve performance. The database should be monitored on a regular basis to look for bottlenecks that affect performance.

Tuning Questions (continued)

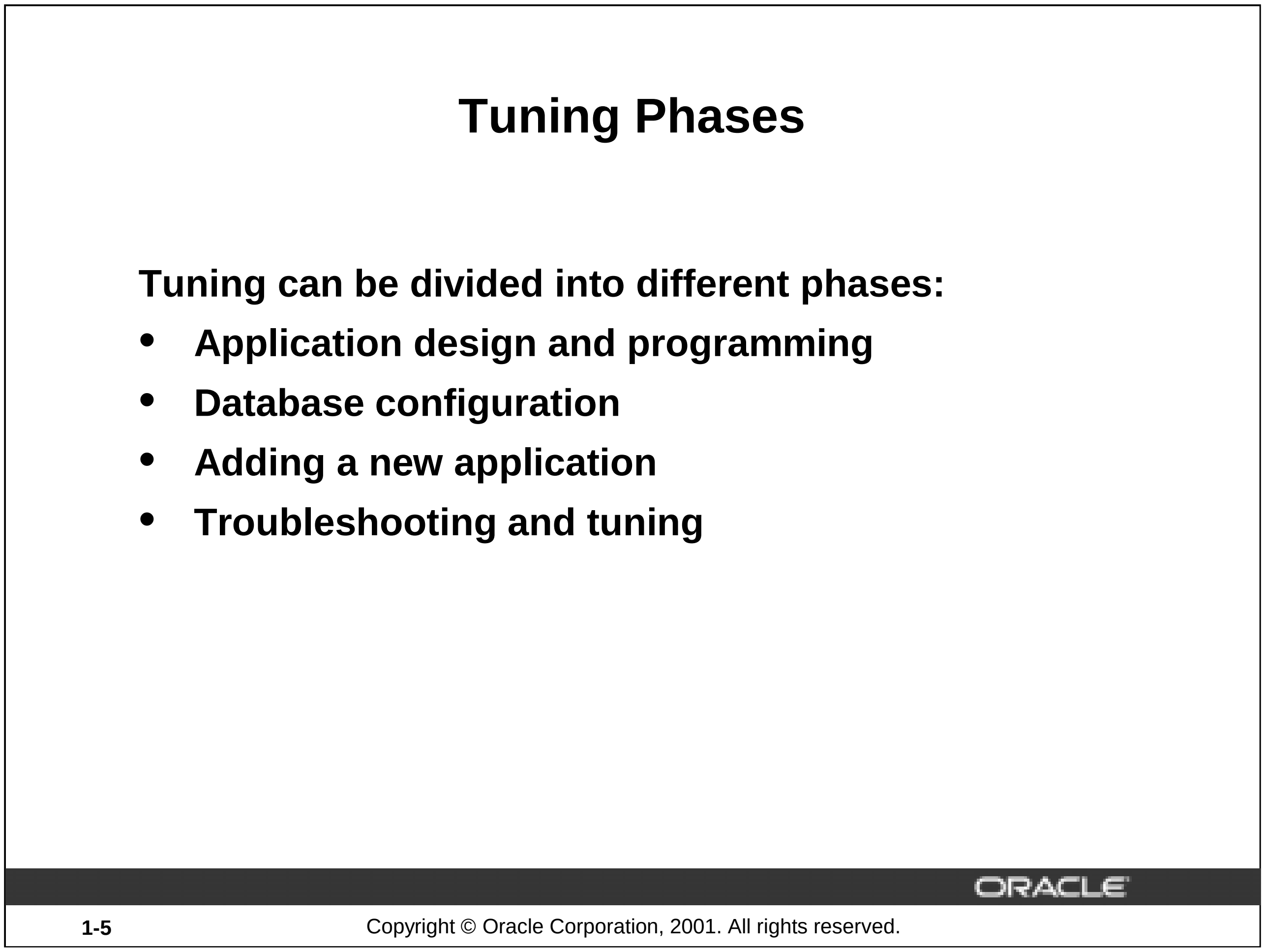
How Much Tuning?

There are basically two forms of tuning:

- Speed: short response time
- High throughput scalability: higher load at a comparable response time or throughput.

During this course, methods to identify and resolve bottlenecks will be discussed. The result of tuning should be visible to users, either as a decrease in the time it takes to perform a task, or as an increase in the number of concurrent sessions.

Tuning is performed because either a problem already exists or the DBA wishes to prevent problems from occurring. Some examples of items to be monitored are critical table growth, changes in the statements users execute, and I/O distribution across devices. This course discusses methods to determine where waits and bottlenecks exist, and how to resolve these problems.



Tuning Phases

Tuning can be divided into different phases:

- Application design and programming
- Database configuration
- Adding a new application
- Troubleshooting and tuning

1-5 Copyright © Oracle Corporation, 2001. All rights reserved.

Tuning Phases

Application Design and Programming

Whenever possible, tuning should start at this level. With a good design, many tuning problems do not occur. For example, although it is normally good practice to fully normalized tables to reduce redundancy, this may result in a high number of table joins. By de-normalizing the performance of the application may improve dramatically.

Database Configuration

It is important to monitor hot spots, even on fast disks, you should plan the data configuration in order to enable faster recovery times, and faster data access.

Adding a New Application

When adding a new application to an existing system, the workload changes. Any major change in the workload should be accompanied by performance monitoring.

Troubleshooting and Tuning

This is the methodology recommended for production databases. It consists of looking for bottlenecks, and resolving them. Use tools to identify performance problems. By examining this data form a hypothesis of what is causing the bottleneck. From the hypothesis develop a solution, and implement it. Run a test load against the database to determine if the performance problem has been resolved.

Tuning Goals

- Reducing or eliminating waits
- Accessing the least number of blocks
- Caching blocks in memory
- Response time
- Throughput
- Load
- Recovery time

ORACLE

Tuning Goals

Your primary goal in tuning an Oracle9*i* database is to make sure that users get responses to their statements as quickly as possible. Because “as quickly as possible” is an ambiguous term, the time measure must be quantified in some manner. The goal is usually measured in terms of response time, throughput, load, or recovery time.

Tuning goals can arise due to a Service Level Agreement. For example, Process A must complete within a specified time period, or a certain number of transactions per second have to be processed.

Examples of Measurable Tuning Goals

- **Improved response time**
- **Improved database availability**
- **Improved database hit percentages**
- **Improved memory utilization**
- **Fewer waits**

Examples of Measurable Tuning Goals

When tuning an Oracle9*i* database environment, the DBA should establish measurable tuning goals. Without them, it will be difficult to determine when enough tuning has been performed.

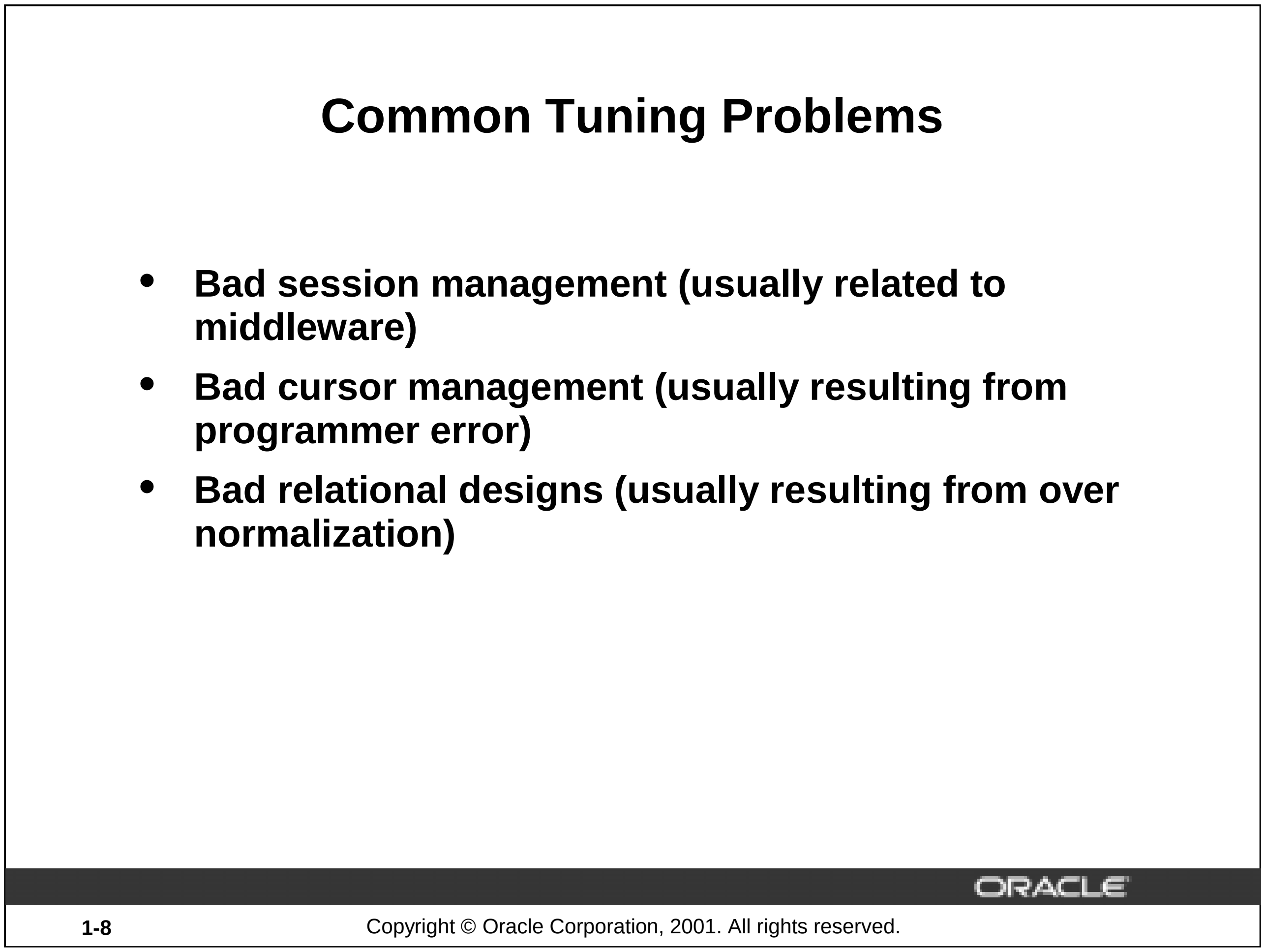
Response time is how long it takes a user to receive data from a request (for example, the result set of a query), or update a table, or generate a report.

Database availability is also a good tuning goal. Availability can be impacted due to backup and recovery, or from shutting down and starting the instance to tune parameters.

Database hit percentages provide a good baseline for determining if performance is increasing or decreasing over time.

Memory utilization is also a valid measure, because excessive paging and swapping can impact database and operating system performance. Memory utilization may also impact database hit percentages.

Checking for waits and bottlenecks is another good method for determining whether performance can be improved.



Common Tuning Problems

- **Bad session management (usually related to middleware)**
- **Bad cursor management (usually resulting from programmer error)**
- **Bad relational designs (usually resulting from over normalization)**

1-8

Common Typing Problems

Bad Session Management
An example of this is a Web page which continually logs on and off a database. This costs the end user time while the logon procedures are followed.

Bad Cursor Management

For example, an application that does not make use of bind variables in the where clause. If `CURSOR_SHARING` is set to SIMILAR or FORCE, then:

(1) `select * from hr.employees where employee_id = 7900;` and

(2) `select * from hr.employees where employee_id = 7369;`

will both parse to a single cursor,

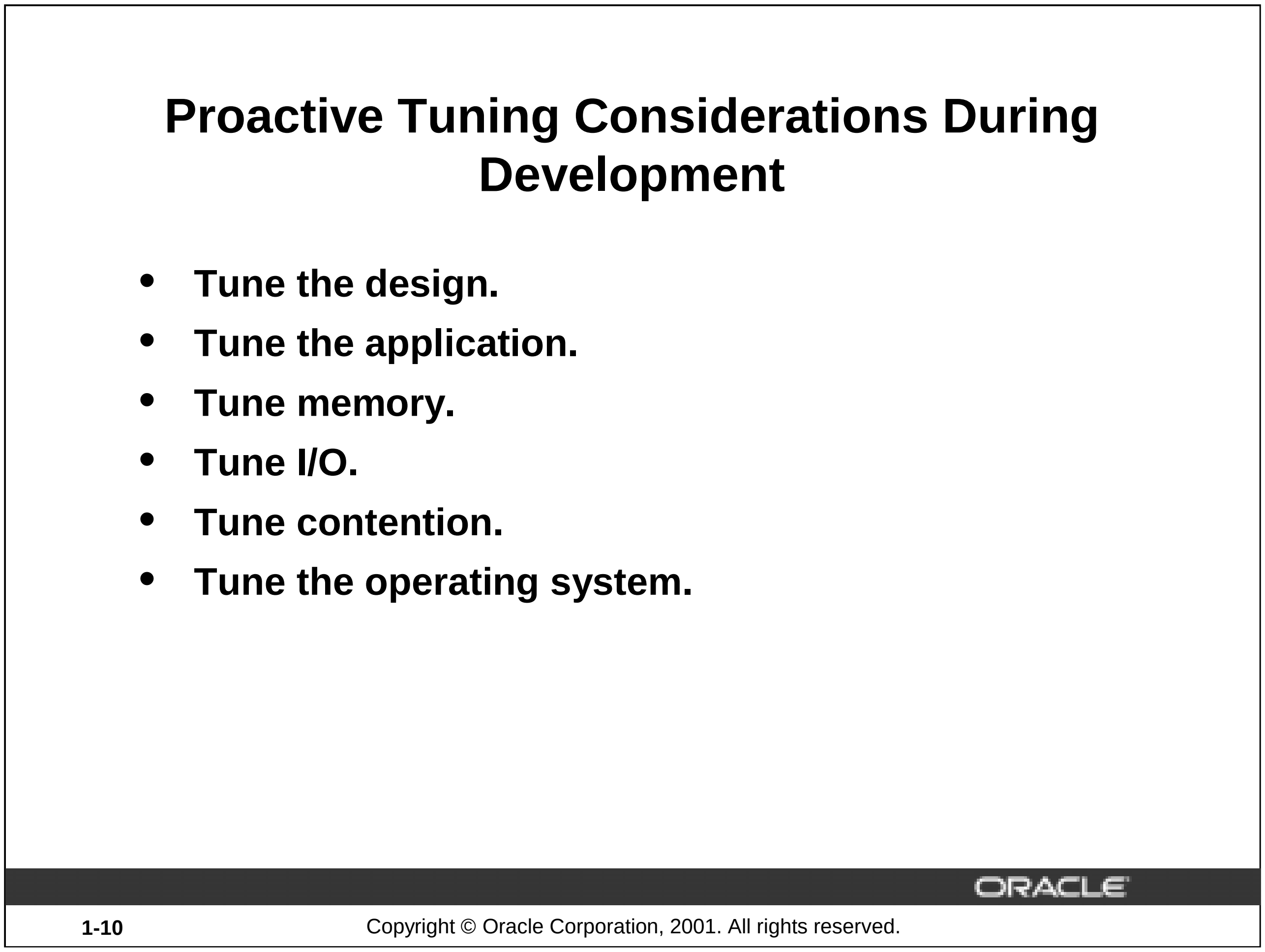
```
select * from hr.employees where employee_id = :SYS_B_0;
```

even though the SQL text is different.

Bad Relational Designs
For example, collecting the wrong columns into tables would require many table joins in order to produce output that could have been obtained from a single table.

Results of Common Tuning Problems

- Bad session management:
  - Limits scalability to a point you cannot exceed
  - Makes the system one or two orders of magnitude slower than it should be
- Bad cursor management makes scalability more limited (not as bad as session management).
- Bad relational design
  - Unnecessary table joins performed
  - Usually a result of trying to build an object interface of relational storage



Proactive Tuning Considerations During Development

- Tune the design.
- Tune the application.
- Tune memory.
- Tune I/O.
- Tune contention.
- Tune the operating system.

1-10

Tuning Steps During Development

During the development of a new system, the following order of tuning implementation is recommended:

1. Design
2. Application
3. Memory
4. Input/output (I/O)
5. Contention
6. Operating system

Repeat the process if your goals have not yet been achieved.

The rationale for this structure is that improvements early in the sequence can save you from having to deal with problems later.

For example, if your applications use many full table scans, this may show up as excessive I/O. However, there is no point in resizing the buffer cache or redistributing disk files, if you can rewrite the queries so that they access only four blocks instead of four thousand.

The first two steps are typically the responsibility of the system architects and application developers; however, the DBA may also be involved in application tuning.

Tuning Steps During Production

- **Locate the bottleneck by using tools.**
- **Determine the reason for the bottleneck.**
- **Resolve the cause.**
- **Check that the bottleneck has been resolved.**

Tuning Steps During Production

The tuning methodology for production systems works by resolving problems when they occur:

1. Locate a bottleneck by using tools, such as STATSPACK, UTLBSTAT and UTLESTAT, or Oracle Enterprise Manager.
2. The bottleneck usually manifests itself as a wait event. Determine the reason for the wait event.
3. Resolve the cause of the wait. This could mean changing the size of a member of the System Global Area.
4. Check that the change has produced a positive effect on the system by running the application again, and then using the tools used in step 1.
5. Repeat the process if your goals have not yet been achieved.

The rationale for this structure is that redoing the same tuning methodology that was used for a development system wastes time. If the system requires a major overhaul, then using the development system methodology is beneficial.

Performance versus Safety Trade-Offs

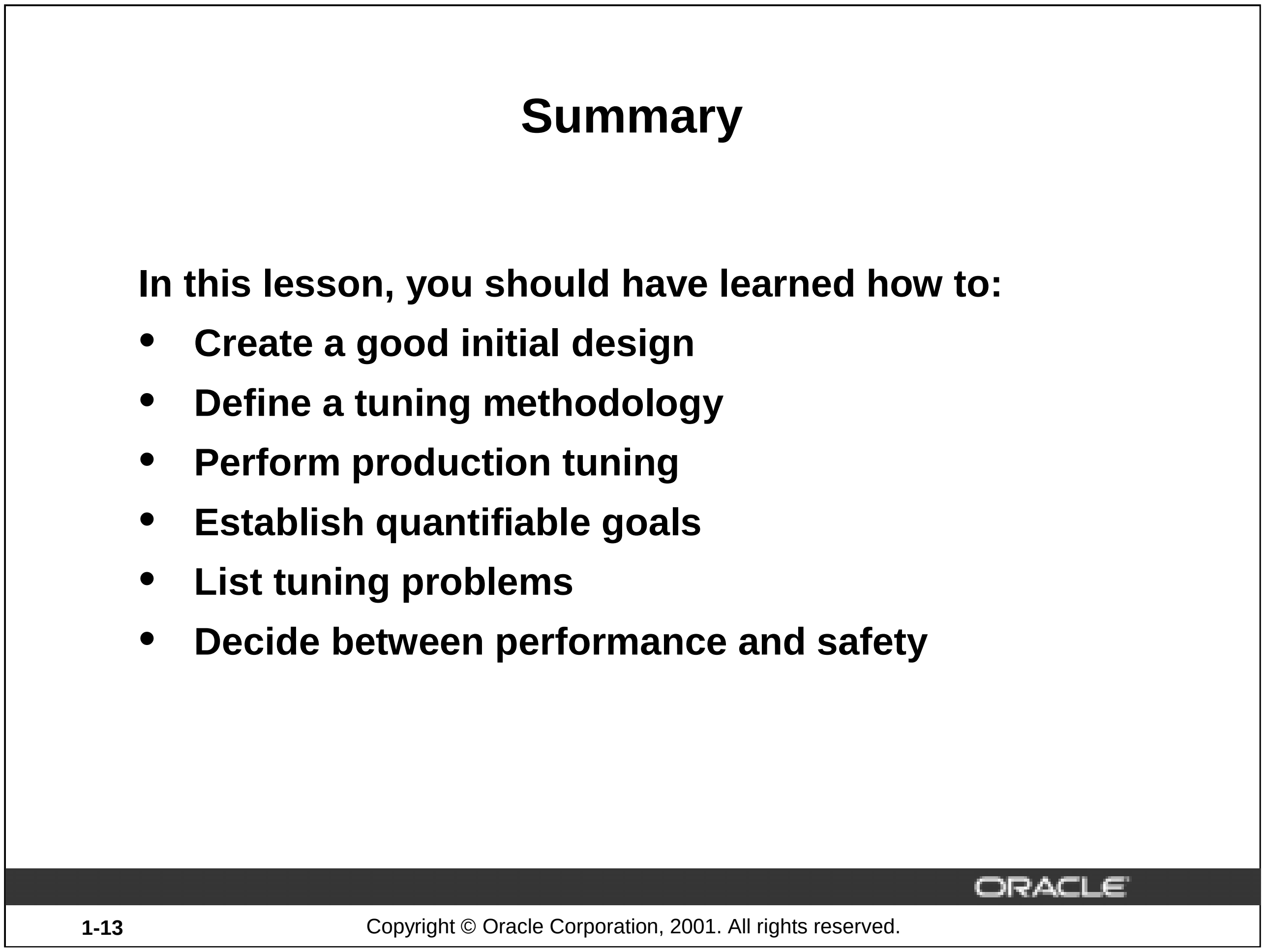
Factors that affect performance:

- **Multiple control files**
- **Multiple redo log members in a group**
- **Frequent checkpointing**
- **Backing up datafiles**
- **Performing archiving**
- **Block check numbers**
- **Number of concurrent users and transactions**

Performance Trade-Offs

There is always a trade-off with performance. In order to increase performance, something else is necessarily affected adversely. Often, recovery time is what suffers. The safer the database administrator makes the database, the slower it runs.

The decision has to be made regarding just how safe to make the database, and what level of performance is required: how many concurrent users must have access to the database, how many transactions per second must take place, and so on.



Summary

In this lesson, you should have learned how to:

- Create a good initial design
- Define a tuning methodology
- Perform production tuning
- Establish quantifiable goals
- List tuning problems
- Decide between performance and safety

1-13

2

Diagnostic and Tuning Tools

Objectives

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

- Describe how the `alert.log` file is used
- Describe how background trace files are used
- Describe how user trace files are used
- Describe the statistics kept in the dynamic performance views
- Collect statistics using STATSPACK
- Describe how STATSPACK collects statistics
- List and describe other tools that can be used during tuning

Maintenance of the Alert Log File

- **The alert log file consists of a chronological log of messages and errors.**
- **Check the alert log file regularly to:**
  - **Detect internal errors (ORA-600) and block corruption errors**
  - **Monitor database operations**
  - **View the nondefault initialization parameters**
- **Remove or trim the file regularly after checking.**

The Alert Log File

It is important for the database administrator to check the alert log file regularly to detect problems before they become serious.

The following information is logged in the alert log file:

- Internal errors (ORA-600) and block corruption errors (ORA-1578 or ORA-1498)
- Operations that affect database structures and parameters, and statements such as `CREATE DATABASE`, `STARTUP`, `SHUTDOWN`, `ARCHIVE LOG`, and `RECOVER`
- The values of all nondefault initialization parameters at the time the instance starts
- The location of the `ALERT.LOG` file is given by the parameter `BACKGROUND_DUMP_DEST`

Tuning Using the Alert Log File

The Alert log file contains the following information which can be used in tuning the database:

- **Checkpoint start and end times**
- **Incomplete checkpoints**
- **Time to perform archiving**
- **Crash recovery start and complete times**

Checkpointing Start and End Times

These values are written into the `ALERT.LOG` file only if the `CHECKPIONTS_TO_ALERT` parameter has been set to true.

Background Processes Trace Files

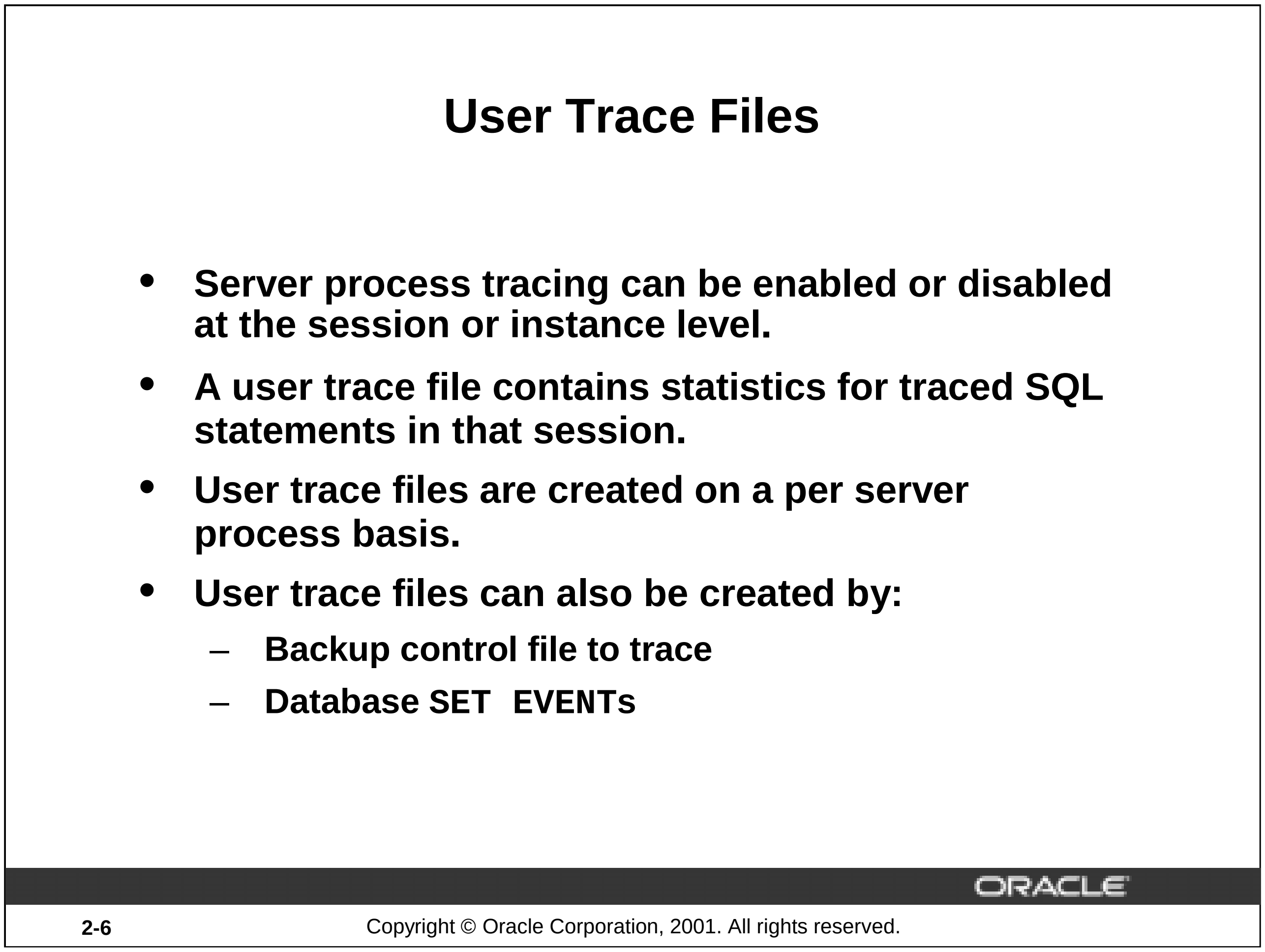
- Oracle server dumps information about errors detected by any background process into trace files.
- Oracle Support uses these trace files to diagnose and troubleshoot.
- These files do not usually contain tuning information.

2-5

Background Processes Trace Files

These files are created by the background processes. In general these files contain diagnostic information, not information regarding performance tuning.

By using events, Oracle Support can write information to these files regarding performance.



User Trace Files

- Server process tracing can be enabled or disabled at the session or instance level.
- A user trace file contains statistics for traced SQL statements in that session.
- User trace files are created on a per server process basis.
- User trace files can also be created by:
  – Backup control file to trace
  – Database `SET EVENTs`

2-6

User Trace Files

User trace files can be generated by server processes at the user's or DBA's request.

Instance-Level Tracing

This trace logging is enabled or disabled by the `SQL_TRACE` initialization parameter.

The default value is FALSE.

Session-Level Tracing

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

```
SQL > EXECUTE dbms_system.set_sql_trace_in_session(8,12,TRUE);
```

where 8 and 12 are the system identifier and serial numbers of the connected user.

The `DBMS_SYSTEM` package is created when the `catproc.sql` script is run. This script is located in the following directory:

`$ORACLE_HOME/rdbms/admin` on UNIX systems, or

`%ORACLE_HOME%\rdbms\admin` for NT.

The following statement enables the writing to a trace file for the session of the connected user:

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

Views, Utilities, and Tools

- **Dynamic troubleshooting, performance, and dictionary views:**
  - **`V$`*`xxx`* dynamic troubleshooting and performance views**
  - **`DBA_` *`xxx`* dictionary views**
- **`utlbstat.sql` and `utlestat.sql` scripts**
- **`STATSPACK`**
- **Oracle wait events**
- **Oracle diagnostics and tuning packs**

Dictionary and Dynamic Views

Oracle server displays all system statistics in the V$SYSSTAT view, and uses many other views for performance and troubleshooting information. You can query these views to find cumulative totals since the instance started but this is often unhelpful; if your instance is rarely shut down, the statistics may cover a long period and have little meaning.

Oracle displays data storage statistics in DBA_*xxx* views that will help you in troubleshooting the segments' storage (tables, clusters, and indexes).

The UTLBSTAT and UTLESTAT Utilities

You should gather performance figures over a defined period, probably your busiest time of day or end of month, and produce a hard-copy report.

You can do this with the `utlbstat.sql` and `utlestat.sql` scripts.

Experienced consultants usually begin a tuning project by using this utility to capture data.

STATSPACK

You can also use STATSPACK to gather statistics. Similar in essence to ULBSTAT and ULTESTAT, STATSPACK also enables a DBA to collect statistics over a period of time and have the statistics stored in the database. This gives the DBA the opportunity to compare statistics over different periods of time.

Oracle Wait Events

If you are troubleshooting, you need to know when a process has waited for any resource. A list of wait events are present in the server.

Some dictionary views display the events for which sessions had to wait.

Oracle Diagnostics and Tuning Packs Applications

You can also use the Oracle GUI tools provided with the Oracle diagnostics and tuning packs, a set of Windows-based applications addressing many Oracle performance management areas, such as graphical monitoring, analysis, and automated tuning of Oracle databases.

Dictionary and Special Views

- **The following dictionary and special views provide useful statistics after executing the `ANALYZE` command:**
  - **`DBA_TABLES`, `DBA_TAB_COLUMNS`**
  - **`DBA_CLUSTERS`**
  - **`DBA_INDEXES`, `INDEX_STATS`**
  - **`INDEX_HISTOGRAM`, `DBA_TAB_HISTOGRAMS`**
- **This statistics information is static until you reexecute the `ANALYZE` command, or run `DBMS_STATS`.**

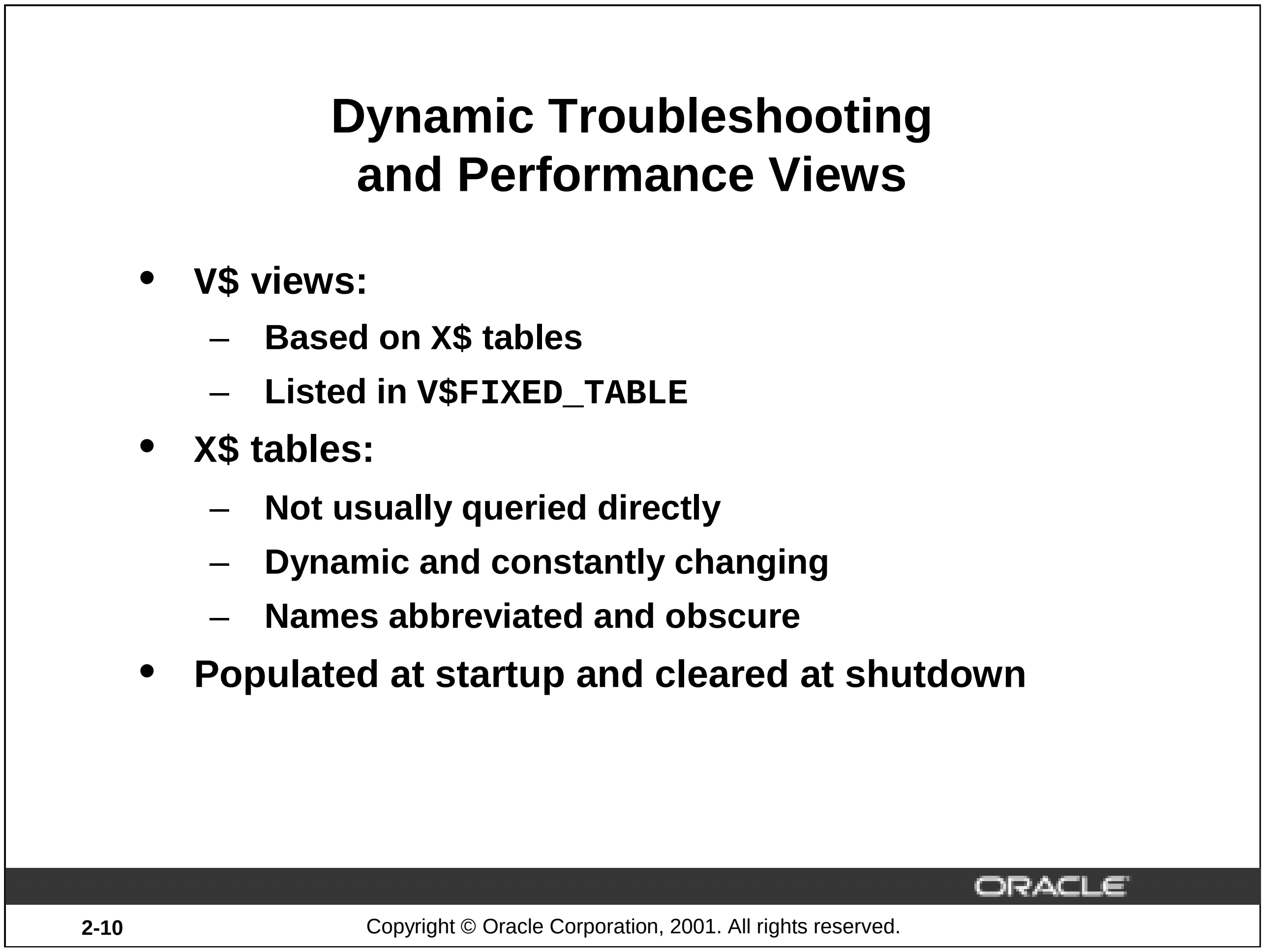
Dictionary and Special Views

When you need to look at data storage in detail, you need to use the `ANALYZE` command, which collects statistics and populates `DBA_`*`xxx`* and special view columns.

`ANALYZE` populates columns in the views concerned with:

- Table data storage within extents and blocks:
  - `DBA_TABLES`
  - `DBA_TAB_COLUMNS`
- Cluster data storage within extents and blocks:
  - `DBA_CLUSTERS`
- Index data storage within extents and blocks, and indexation usefulness:
  - `DBA_INDEXES`
  - `INDEX_STATS`
- Nonindexed and indexed columns data distribution:
  - `DBA_TAB_HISTOGRAMS`
  - `INDEX_HISTOGRAM`

This command is described in more detail in a later lesson, “Using Oracle Blocks Efficiently.”



Dynamic Troubleshooting and Performance Views

- **V$ views:**
  - **Based on X$ tables**
  - **Listed in `V$FIXED_TABLE`**
- **X$ tables:**
  - **Not usually queried directly**
  - **Dynamic and constantly changing**
  - **Names abbreviated and obscure**
- **Populated at startup and cleared at shutdown**

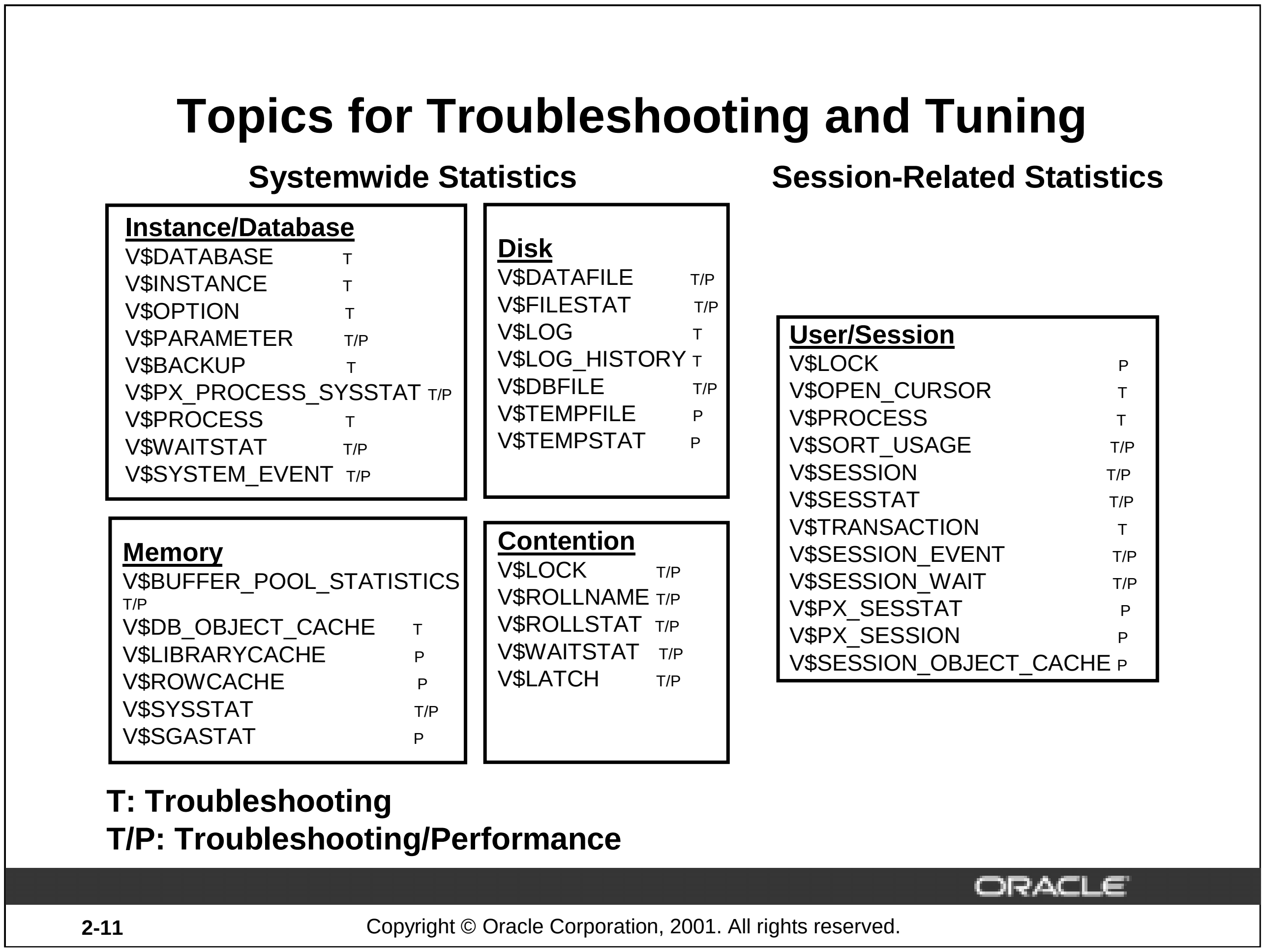
V$ Views

- These are based on `X$` tables—memory structures which hold instance information and are therefore available when the instance is in a NOMOUNT or MOUNT state.
- They are listed in `V$FIXED_TABLE`.
- The V$ views (actually synonyms for V_$ views) belong to the `sys` user.

X$ Tables

- These are not usually queried directly; not all the information is necessarily useful, and column names tend to be abbreviated and obscure.
- The `X$` tables are dynamic, and their contents are constantly changing.
- The `V$` views, and the underlying `X$` tables, are populated at instance startup and cleared out at shutdown.
- They hold timing information if you set the `TIMED_STATISTICS` `init.ora` parameter to TRUE or if you execute the following SQL command:

```
    SQL> ALTER SYSTEM SET timed_statistics = TRUE;
```



Topics for Troubleshooting and Tuning

Systemwide Statistics

Instance/Database	
V$DATABASE	T
V$INSTANCE	T
V$OPTION	T
V$PARAMETER	T/P
V$BACKUP	T
V$PX_PROCESS_SYSSTAT	T/P
V$PROCESS	T
V$WAITSTAT	T/P
V$SYSTEM_EVENT	T/P

Disk	
V$DATAFILE	T/P
V$FILESTAT	T/P
V$LOG	T
V$LOG_HISTORY	T
V$DBFILE	T/P
V$TEMPFILE	P
V$TEMPSTAT	P

Memory	
V$BUFFER_POOL_STATISTICS	T/P
V$DB_OBJECT_CACHE	T
V$LIBRARYCACHE	P
V$ROWCACHE	P
V$SYSSTAT	T/P
V$SGASTAT	P

Contention	
V$LOCK	T/P
V$ROLLNAME	T/P
V$ROLLSTAT	T/P
V$WAITSTAT	T/P
V$LATCH	T/P

Session-Related Statistics

User/Session	
V$LOCK	P
V$OPEN_CURSOR	T
V$PROCESS	T
V$SORT_USAGE	T/P
V$SESSION	T/P
V$SESSTAT	T/P
V$TRANSACTION	T
V$SESSION_EVENT	T/P
V$SESSION_WAIT	T/P
V$PX_SESSTAT	P
V$PX_SESSION	P
V$SESSION_OBJECT_CACHE	P

T: Troubleshooting
T/P: Troubleshooting/Performance

2-11

Systemwide Statistics

Views Pertaining to the Instance/Database

- `V$PX_PROCESS_SYSSTAT`: Parallel query system statistics
- `V$PROCESS`: Information about currently active processes
- `V$WAITSTAT`: Contention statistics
- `V$SYSTEM_EVENT`: Total waits for particular events

Views Pertaining to Memory

- `V$BUFFER_POOL_STATISTICS`: Buffer pools allocation on the instance (created `by the $ORACLE_HOME/rdbms/admin/catperf.sql` script)
- `V$DB_OBJECT_CACHE`: Database objects cached in the library cache
- `V$LIBRARYCACHE`: Library cache performance and activity statistics
- `V$ROWCACHE`: Data dictionary hits and misses activity
- `V$SYSSTAT`: Basic instance statistics

Systemwide Statistics (continued)

Views Pertaining to Disk Performance

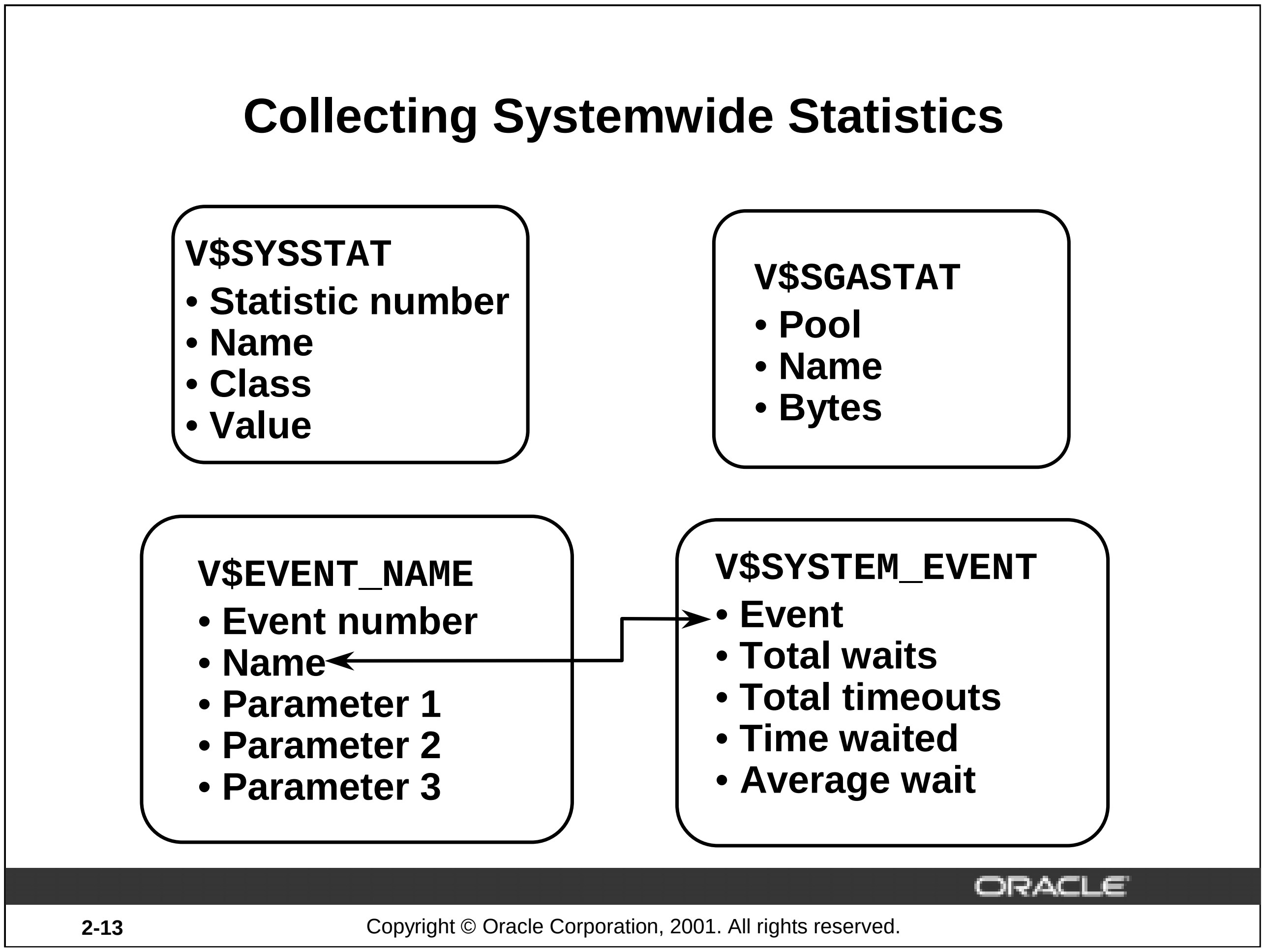
- `V$FILESTAT`: Data file read/write statistics
- `V$TEMPSTAT`: Information about file read/write statistics for temporary tablespace data files

Views Pertaining to Contention

- `V$LATCH`: Statistics for each type of latch
- `V$ROLLSTAT`: Statistics for all online rollback segments
- `V$WAITSTAT`: Block contention statistics (the `TIMED_STATISTICS` init.ora parameter should be set to TRUE)

Session-Related Statistics

- `V$LOCK`: Locks currently held by the server and outstanding requests for a lock or latch
- `V$OPEN_CURSOR`: Cursors currently opened and parsed by each session
- `V$SORT_USAGE`: Size of temporary segments and sessions creating them; identification of processes doing disk sorts
- `V$SESSTAT`: User session statistics
- `V$SESSION_EVENT`: Information on waits for an event by a session
- `V$SESSION_WAIT`: Resources or events for which active sessions are waiting
- `V$PX_SESSTAT`: Information about the sessions running in parallel.



Collecting Systemwide Statistics

General Systemwide Statistics

All kinds of systemwide statistics are cataloged in the `V$STATNAME` view: about 286 statistics are available.

The server displays all calculated system statistics in the `V$SYSSTAT` view. You can query this view to find cumulative totals since the instance started.

Example:

```
SQL>  SELECT name,class,value FROM v$sysstat;
NAME                       CLASS               VALUE
------------------        ------           ----------
logons cumulative              1                 6393
logons current                 1                   10
opened cursors cumulative      1               101298
table scans (short tables)    64                 6943
table scans (long tables)     64                  344
table scans (rowid ranges)    64                    0
redo entries                   2              1126226
redo size                      2            816992940
redo buffer allocation
retries                        2                 1461
redo wastage                   2              5874784
……
The results shown are only a partial display of the output.
```

General Systemwide Statistics

These statistics are classified by the topics of tuning:

- Class 1 refers to general instance activity.
- Class 2 refers to redo log buffer activity.
- Class 4 refers to locking.
- Class 8 refers to database buffer cache activity.
- Class 16 refers to OS activity.
- Class 32 refers to parallelization.
- Class 64 refers to tables access.
- Class 128 refers to debugging purposes.

SGA Global Statistics

Server displays all calculated memory statistics in the `V$SGASTAT` view. You can query this view to find cumulative totals of detailed SGA usage since the instance started.

Example

```
SQL>     SELECT * FROM v$sgastat;
POOL                NAME                             BYTES
------              -------------------------        ----------
                    fixed_sga                        46136
                    db_block_buffers                 409600
                    log_buffer                       524288
shared pool         free memory                      8341616
shared pool         SYSTEM PARAMETERS                42496
shared pool         transaction                      64800
shared pool         dictionary cache                 156524
shared pool         library cache                    358660
shared pool         sql area                         551488
```

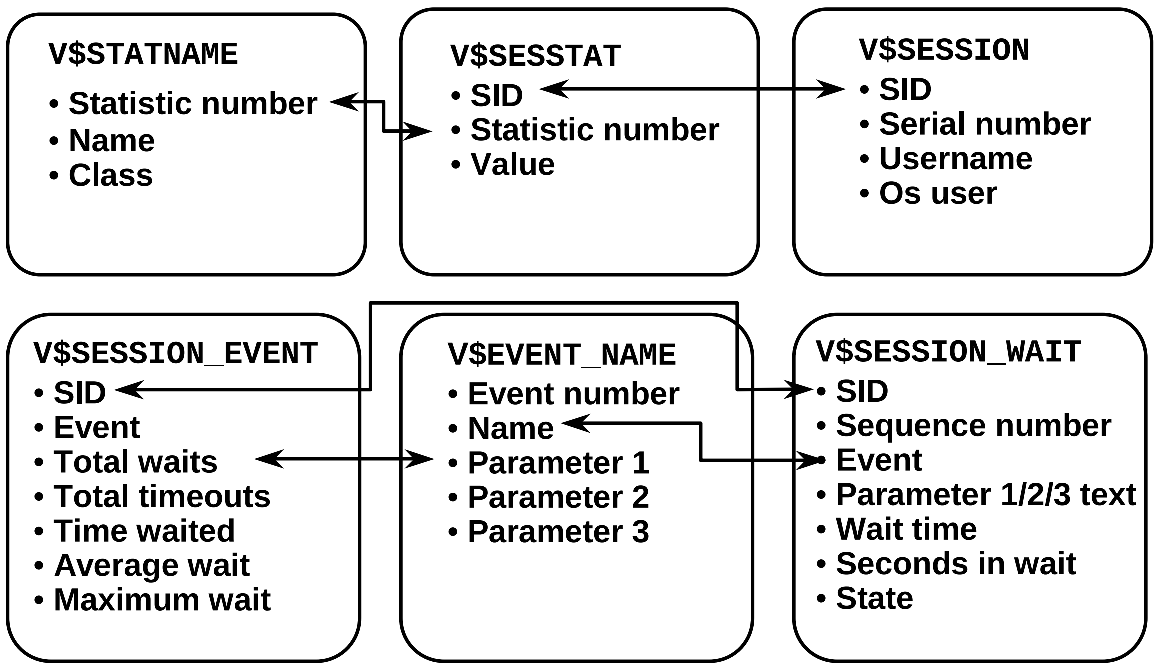
Waiting Events Statistics

All kinds of waiting events are cataloged in `V$EVENT_NAME` view: about 286 events are available.

Cumulative statistics for all sessions are stored in `V$SYSTEM_EVENT`, which shows the total waits for a particular event since instance startup.

If you are troubleshooting, you need to know when a process has waited for any resource.

Collecting Session-Related Statistics



General Session-Related Statistics

Session data is cumulative from connect time.

You can display current session information for each user logged on.

The V$MYSTAT view displays the statistics of the current session.

Example: Determine the type of connection the users have.

```
SQL>    SELECT sid, username, type, server FROM v$session;
SID    USERNAME         TYPE                SERVER
-----  ------------     -----------         ----------------
  1                     BACKGROUND          DEDICATED
  2                     BACKGROUND          DEDICATED
  3                     BACKGROUND          DEDICATED
  4                     BACKGROUND          DEDICATED
  5                     BACKGROUND          DEDICATED
  6                     BACKGROUND          DEDICATED
  9    SYSTEM           USER                DEDICATED
 10    HR               USER                NONE
 11    SH               USER                SHARED
```

Oracle server displays all calculated session statistics in the V$SESSTAT view. You can query this view to find session cumulative totals since the instance started.

General Session-Related Statistics (continued)

Example: Determine the sessions that consume more than 30,000 bytes of PGA memory.

```
SQL> select username,name,value
2  from v$statname n, v$session s, v$sesstat t
3  where s.sid=t.sid
4  and   n.statistic#=t.statistic#
5  and   s.type='USER'
6  and   s.username is not null
7  and   n.name='session pga memory'
8* and   t.value > 30000;
USERNAME    NAME                   VALUE
----------  -------------------    -----------
SYSTEM      session pga memory     468816
```

Session Waiting Events Statistics

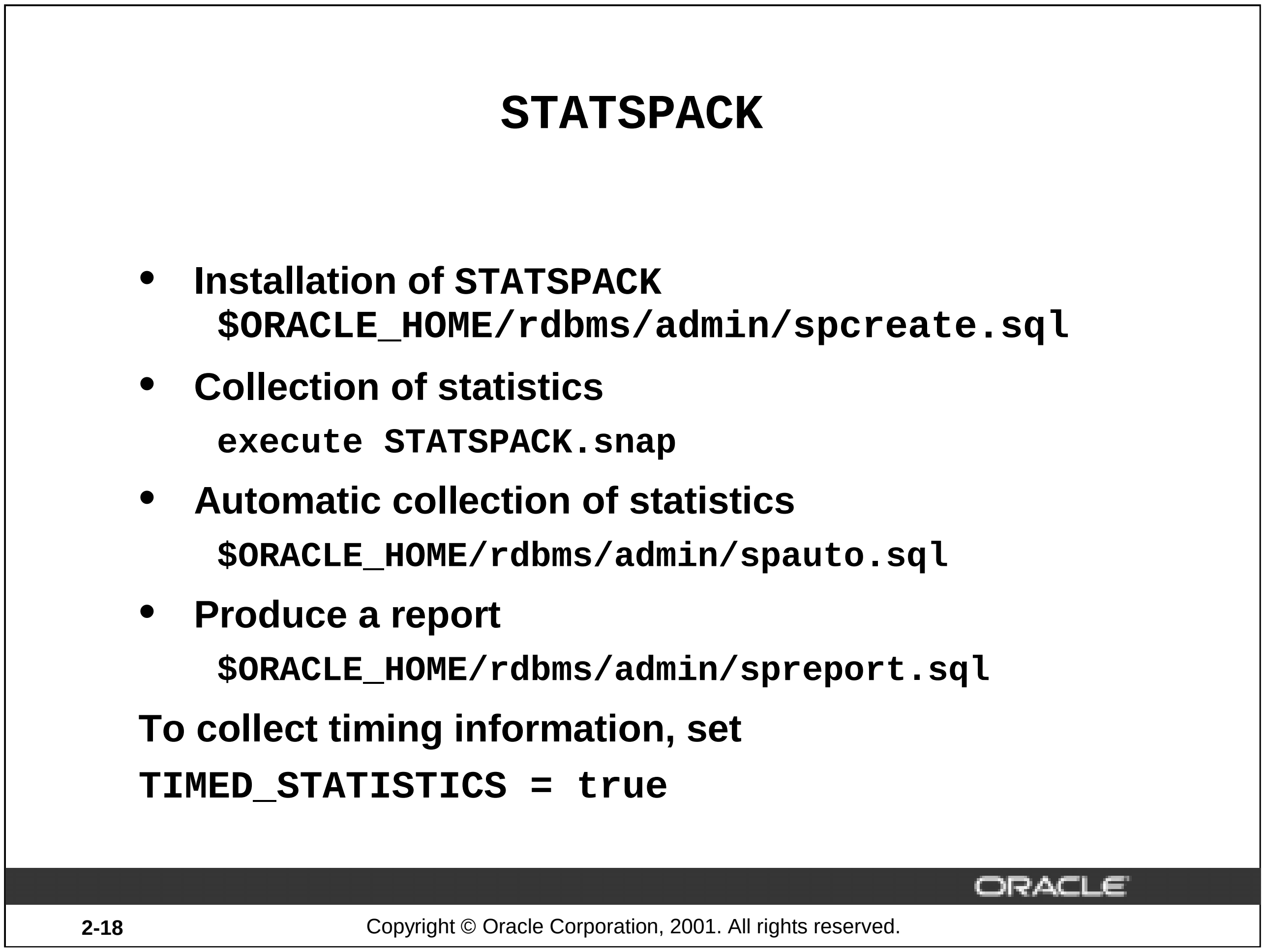
`V$SESSION_EVENT` shows, by session, the total waits for a particular event since instance startup.

`V$SESSION_WAIT` view lists the resources or events for which active sessions are waiting.

If you are troubleshooting, you need to know when a process has waited for any resource. The structure of `V$SESSION_WAIT` makes it easy to check in real time whether any sessions are waiting, and why.

```
 SQL> select sid, event
 2  from V$SESSION_WAIT
 3* where wait_time = 0;
SID         EVENT
----------- ---------------------------------------
   1        pmon timer
   2        rdbms ipc message
   3        rdbms ipc message
   9        rdbms ipc message
   16       rdbms ipc message
   17       rdbms ipc message
   10       rdbms ipc message
   4        rdbms ipc message
   5        smon timer
9  rows selected.
```

You can then investigate further to see whether such waits occur frequently and whether they can be correlated with other phenomena, such as the use of particular modules.



STATSPACK

- **Installation of STATSPACK**

  `$ORACLE_HOME/rdbms/admin/spcreate.sql`
- **Collection of statistics**

  `execute STATSPACK.snap`
- **Automatic collection of statistics**

  `$ORACLE_HOME/rdbms/admin/spauto.sql`
- **Produce a report**

  `$ORACLE_HOME/rdbms/admin/spreport.sql`

To collect timing information, set

`TIMED_STATISTICS = true`

2-18

`STATSPACK`

The `STATSPACK` package has been available with Oracle Database from Oracle8.1.6. When initially installed roughly 80 MBs of the users default tablespace is used. This may grow later with the tables storing snapshot information.

Installation of the STATSPACK Package

Installing the `STATSPACK` utility creates the `Perfstat` user, who owns all PL/SQL code and database objects created (including the `STATSPACK` tables, the constraints and the `STATSPACK` package). During the installation you will be prompted for the `Perfstat` user's default and temporary `tablespaces`.

Collecting Statistics

Take a snapshot of performance data log in to SQL*Plus as the `Perfstat` user, and execute the `STATSPACK.snap` procedure. This stores the current values for the performance statistics in the `STATSPACK` tables, which can be used as a baseline snapshot for comparison with another snapshot taken at a later time.

Automatically Collecting Statistics

To compare performance from one day, week, or year to the next, there must be multiple snapshots taken over a period of time. The best method to gather snapshots is to automate the collection at regular time intervals.

The `spauto.sql script` makes use of `dbms_job`, the automated method for collecting statistics. The supplied script schedules a snapshot every hour, on the hour.This can be changed to suit the requirements of the system.

Producing a Report

To examine the change in statistics between two time periods, execute the `spreport.sql` file while being connected to the `Perfstat` user. The user is shown a list of collection periods and selects a start and end period. The difference between the statistics at each end point is then calculated and put into a file named by the user.

STATSPACK Output

Information found on the first page:

- **Database and instance name**
- **Time at which the snapshots were taken**
- **Current sizes of the caches**
- **Load profile**
- **Efficiency percentages of the instance**
- **Top five wait events**

First Page of the STATSPACK Report

The first page summarizes the report. The items found on this page includes most of the information that the DBA requires to enhance performance. Each of these areas will be covered in greater depth during the course.

Cache Sizes

The current size of the buffer cache, shared pool and log buffer are shown here in KB. Also included here is the value of the primary block size.

Load Profile

One value given here is per second, and the other per transaction. What is shown includes the redo size, and physical reads, and physical writes performed during the snapshot period.

Instance Efficiency Percentages

The items listed here are ones that are most often used for tuning a database. The goal is to have all percentages at 100%, or as close as possible. Compare these values with what the normal values are for the database.

Top 5 Wait Events

The full list of wait events appears later in the report, and will be dealt with later in the course. The list given here provides the DBA with the top contention items.

STATSPACK Output

Information found in the remainder of the document:

- **Complete list of wait events**
- **Information on SQL statements currently in the pool**
- **Instance activity statistics**
- **Tablespace and file I/O**
- **Buffer pool statistics**

2-21

STATSPACK Output

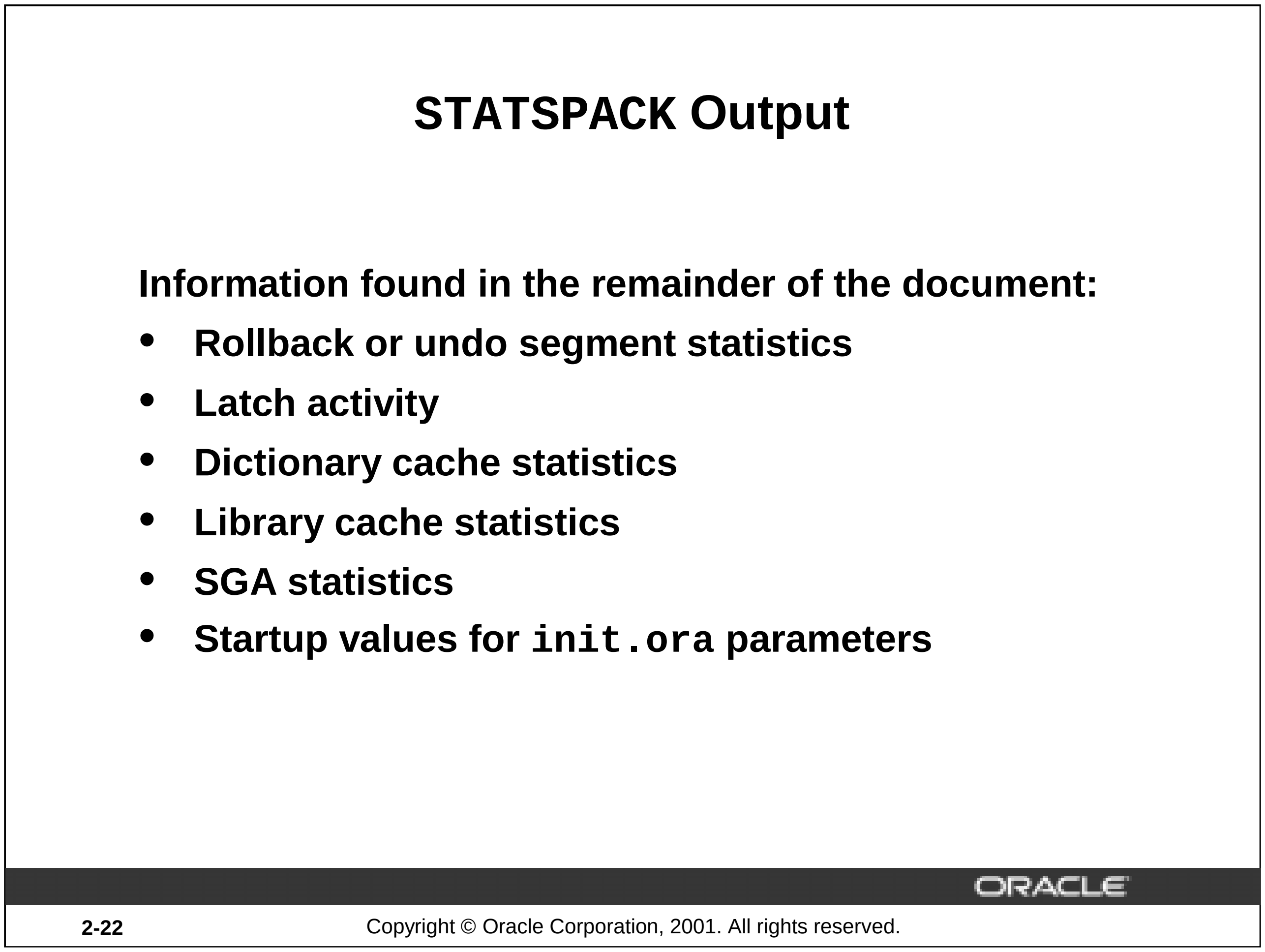
The report generated by `STATSPACK` is similar to that of `utlbstat` and `utlestat`, but there are many areas in which `STATSPACK` is better. The first page summary found with `STATSPACK` aids the DBA, because there is now no need to search through the report to determine these important numbers.

`STATSPACK` also assists in giving information about the SQL statements that are stored in the shared SQL area, thus giving the DBA or developer better information about which statements to tune in order to best use his or her time.

Wait events are also ordered by time so as to put the most problematic events at the top. `STATSPACK` also attempts to put those wait events which are not a problem (for example, `pmon timer`) at the end of the list, regardless of what the time value is.

SQL Statements

Several different ordered lists are given. SQL statements are sorted in order of number of executions, number of parse calls, number of buffer gets, number of reads. By ordering the SQL statements by these different columns, it is easier to determine which statements are causing the heavy work load. These statements should then be tuned first, so as to get the highest return on time spent.



STATSPACK Output

Information found in the remainder of the document:

- **Rollback or undo segment statistics**
- **Latch activity**
- **Dictionary cache statistics**
- **Library cache statistics**
- **SGA statistics**
- **Startup values for `init.ora` parameters**

2-22

Rollback or Undo Segment Statistics

Because the DBA can use either rollback segments or undo segments, STATSPACK covers both options

SGA Statistics

This is made up of two lists, a memory summary, and a breakdown of each area.

Startup Values for `init.ora` Parameters

Because many `init.ora` parameters are dynamic, that is, can be changed without stopping and starting the database, there is no guarantee that the current value will be the one used if the system is restarted.

UTLBSTAT and UTLESTAT Utilities

These utilities:

- **Gather performance figures over a defined period**
- **Produce a hard-copy report**

To use these:

- **Set `TIMED_STATISTICS` to** TRUE
- **Use the `utlbstat.sql` and `utlestat.sql` scripts**
- **Run the scripts from SQL*Plus connected as `SYSDBA`**

`STATSPACK` provides clearer statistics.

Begin and End Scripts

The dynamic views display cumulative totals since the instance started, but this is rarely helpful. If your instance is infrequently shut down, the statistics may cover a long period and have little meaning.

You should gather performance figures over a defined period, probably your busiest time of day or month end, and produce a hard-copy report.

You can do this by running the `utlbstat.sql` script at the beginning of the defined period and then `utlestat.sql` at the end of the period. These scripts are both stored in the `$ORACLE_HOME/rdbms/admin` directory on UNIX and in `%ORACLE_HOME%\Rdbms\Admin` on Windows NT.

The `ultestat.sql` script also generates a hard copy report in the current directory. This report contains statistics for the collection period, gathered by looking at the differences between the beginning and end statistics.

Oracle Wait Events

- **A collection of wait events provides information on the sessions that had to wait or must wait for different reasons.**
- **These events are listed in the `V$EVENT_NAME` view which has the following columns:**
  - **`EVENT#`**
  - **`NAME`**
  - **`PARAMETER1`**
  - **`PARAMETER2`**
  - **`PARAMETER3`**

List of Events

There are more than 286 wait events in the Oracle server, including:

- Free Buffer Wait
- Latch Free
- Buffer Busy Waits
- Db File Sequential Read
- Db File Scattered Read
- Db File Parallel Write
- Undo Segment Tx Slot
- Undo Segment Extension

For the complete list, refer to the *Oracle9i Reference,* Release 9.0.1, Appendix A.

The V$EVENT_NAME View

```
SQL> SELECT name, parameter1, parameter2, parameter3
  2  FROM v$event_name;
```

```
NAME                            PARAMETER1  PARAMETER2  PARAMETER3
------------------------------  ----------  ----------  ----------
PL/SQL lock timer               duration
alter system set mts_dispatcher waited
buffer busy waits               file#       block#      id
library cache pin               handle addr pin address 0*mode+name
log buffer space
log file switch
(checkpoint incomplete)
transaction                     undo seg#   wrap#       count
...
286 rows selected.
```

Parameters Describing a Wait Event

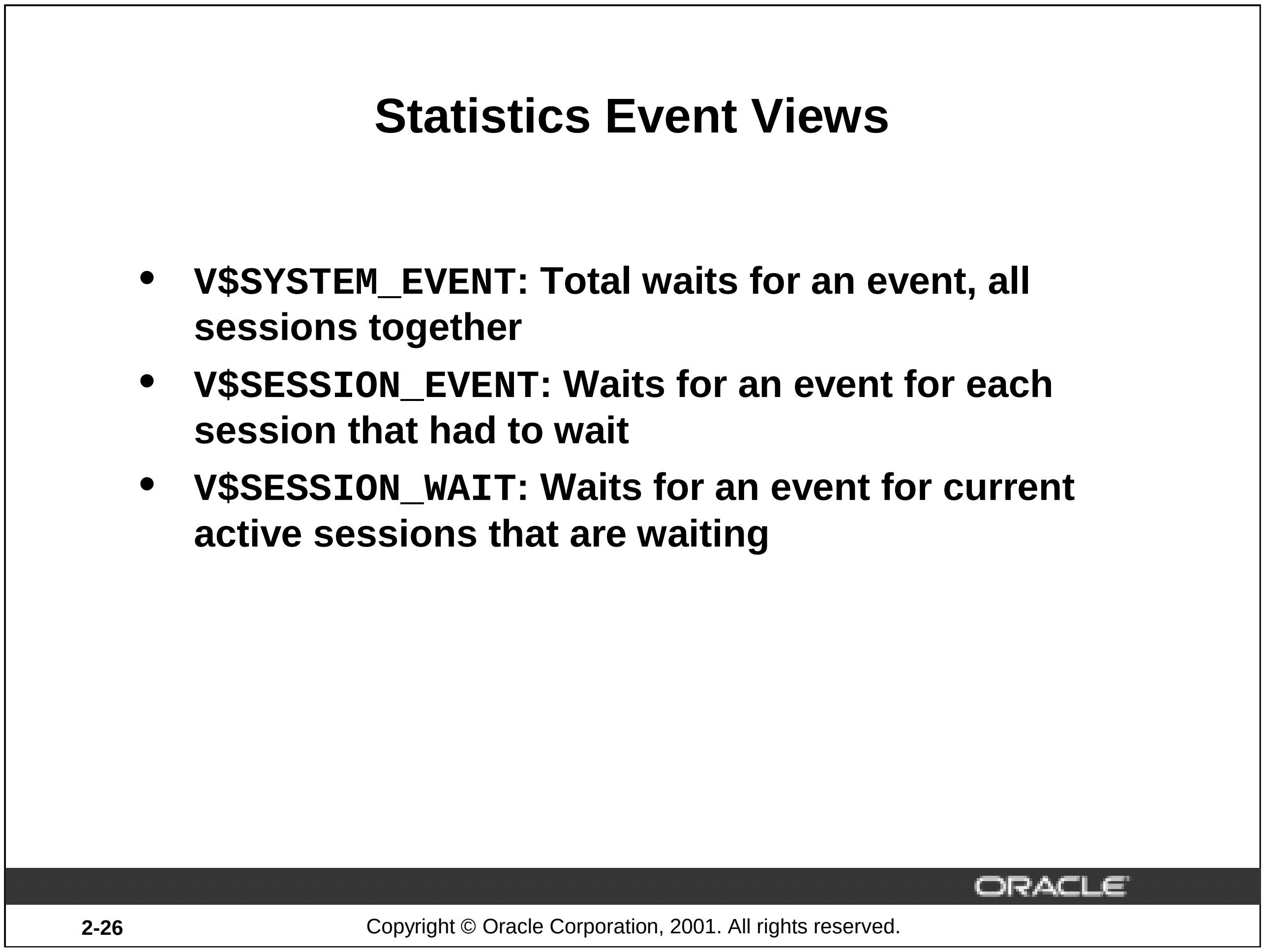
Example 1: The Buffer Busy Waits event waits until a buffer becomes available. This event can be caused by a number of conditions but generally it is when a block must be brought into the buffer cache and a wait is incurrent or a block in the cache needs to modified by a session and another session has been locked.

This event is completed with three parameters:

- FILE# and BLOCK#: These parameters identify the block number in the data file that is identified by the file number for the block for which the server needs to wait.
- ID: The buffer busy wait event is called from different places in the session. Each place in the kernel points to a different reason. ID refers to the place in the session calling this event.

Example 2: The Log File Switch (Checkpoint Incomplete) waits for a log switch because the session cannot wrap into the next log. Wrapping cannot be performed because the checkpoint for that log has not completed.

This event has no parameter.



Statistics Event Views

- **`V$SYSTEM_EVENT`: Total waits for an event, all sessions together**
- **`V$SESSION_EVENT`: Waits for an event for each session that had to wait**
- **`V$SESSION_WAIT`: Waits for an event for current active sessions that are waiting**

Statistics for Waiting Sessions

The statistics results of the sessions that had to wait or are currently waiting for a resource are stored in the `V$SESSION_EVENT` and `V$SESSION_WAIT` views.

Cumulative statistics for all sessions are stored in `V$SYSTEM_EVENT`.

The V$SYSTEM_EVENT View

```
SQL> SELECT event, total_waits, total_timeouts,
  2  time_waited, average_wait
  3  FROM v$system_event;
```

```
EVENT                 TOTAL_    TOTAL_    TIME_      AVERAGE_
                       WAITS  TIMEOUTS   WAITED          WAIT
----------------    ------  --------   ------    ----------
latch free                5         5        5             1
pmon timer              932       535   254430    272.993562
process startup           3                  8    2.66666667
buffer busy waits        12         0        5             5
...
34 rows selected.
```

The V$SYSTEM_EVENT View

V$SYSTEM_EVENT shows the total waits for a particular event since instance startup.

If you are troubleshooting, you need to know when a process has waited for any resource. Therefore, it becomes useful to query this view each time the system slows down.

V$SYSTEM_EVENT contains the following columns:

- EVENT: Name of the wait event
- TOTAL_WAITS: Total number of waits for event
- TOTAL_TIMEOUTS: Total number of timeouts for event
- TIME_WAITED: Total amount of time waited for this event, in hundredths of a second
- AVERAGE_WAIT: The average amount of time waited for this event, in hundredths of a second

The V$SESSION_EVENT View

```
SQL> select sid, event, total_waits,average_wait
  2> from v$session_event where sid=10;
```

```
 SID EVENT                          TOTAL_WAITS AVERAGE_WAIT
---- ------------------------------ ----------- ------------
  10 buffer busy waits                       12            5
  10 db file sequential read                129            0
  10 file open                                1            0
  10 SQL*Net message to client               77            0
  10 SQL*Net more data to client              2            0
  10 SQL*Net message from client             76            0
```

The V$SESSION_EVENT View

V$SESSION_EVENT shows the same information by session. It includes the columns listed in the previous page, with an extra column for SID to identify the session. You can join the SID column to V$SESSION.SID to find user details.

Note: You can query these views directly to find out about all system waits since startup.

The V$SESSION_WAIT View

```
SQL>  SELECT sid, seq#, event, wait_time, state
  2     FROM v$session_wait;
```

```
SID   SEQ#   EVENT                              WAIT   STATE
                                                TIME
---- ------ --------------------------       -----    -------
  1   1284   pmon timer                          0    WAITING
  2   1697   rdbms ipc message                   0    WAITING
  3    183   rdbms ipc message                   0    WAITING
  4   4688   rdbms ipc message                   0    WAITING
  5    114   smon timer                          0    WAITING
  6     14   SQL*Net message from client        -1    WAITED
                                                      SHORT
                                                      TIME
```

The V$SESSION_WAIT View

This view lists the resources or events for which active sessions are waiting.

Columns

- SID: Session identifier
- SEQ#: Sequence number identifying the wait
- EVENT: Resource or event waited for
- P1TEXT: Description of first additional parameter, which corresponds to the PARAMETER1 described for the V$EVENT_NAME view
- P1: First additional parameter value
- P1RAW: First additional parameter value, in hexadecimal
- P2TEXT: Description of second additional parameter, which corresponds to the PARAMETER2 described for the V$EVENT_NAME view
- P2: Second additional parameter value
- P2RAW: Second additional parameter value in hexadecimal
- P3TEXT: Description of third additional parameter, which corresponds to the PARAMETER3 described for the V$EVENT_NAME view
- P3: Third additional parameter value

V$SESSION_WAIT View (continued)

Columns (continued)

- P3RAW: Third additional parameter value, in hexadecimal
- WAIT_TIME

Value	Explanation
>0	The session's last wait time
=0	The session is currently waiting
=-1	The value was less than 1/100 of a second
=-2	The system cannot provide timing information

- SECONDS_IN_WAIT: Number of seconds the event waited
- STATE: Waiting, Waited Unknown Time, Waited Short Time (less than one one-hundredth of a second), Waited Known Time (the value is stored in the WAIT_TIME column)

Note: Not all of the parameter columns are used for all events.

The TIMED_STATISTICS Initialization Parameter

Set the TIMED_STATISTICS parameter to TRUE to retrieve values in the WAIT_TIME column. It is a dynamic initialization parameter.

Performance Manager

- Predefined scopes of statistics:
  - I/O
  - Contention
  - Database instance
  - Load
  - Memory
  - Top resource consumers
  - Overview of performance
    - Overview of cache utilization
    - Overview of user activity
    - Overview of throughput
    - Overview of performance default chart
- User-defined charts

ORACLE

2-31

Performance Manager

Performance Manager is not a part of the basic Oracle Enterprise Manager that is shipped with the database. Rather it is a part of an optional package.

Performance Manager Characteristics

The Performance Manager application captures, computes, and presents performance data in a graphical, real-time view that allows you to monitor the key metrics required to:

- Use memory effectively
- Minimize disk I/O
- Avoid resource contention

The data displayed in real-time mode can be recorded for replay.

You can also define new charts and display windows containing charts in many categories.

Predefined Charts

Seven different categories of predefined charts are available for display. Each category has a set of specific charts that focus on the parent subject.

I/O

These charts include File I/O Rate, File I/O Rate Details, Network I/O Rate, and System I/O Rate.

Performance Manager (continued)

Contention

These charts include Circuit, Dispatcher, Free List Hit %, Latch, Lock, Queue, Redo Allocation Hit %, Rollback NoWait Hit %, and Shared Server.

Database_Instance

These charts include Process, Session, System Statistics, Table Access, Tablespace, Tablespace Free Space, # Users Active, # Users Logged on, # Users Waiting, # Users Waiting for Locks, and # Users Running.

Load

These charts include Buffer Gets Rate, Network Bytes Rate, Redo Statistics Rate, Sort Rows Rate, Table Scan Rows Rate, and Throughput Rate.

Memory

These charts include Buffer Cache Hit %, Data Dictionary Cache Hit %, Library Cache Hit %, Library Cache Details, SQL Area, Memory Allocated, Memory Sort Hit %, Parse Ratio, and Read Consistency Hit %.

Top Resource Consumers

Top Resource Consumers is one of the predefined charts for database services.

Overview of Performance

The overview displays a composite of the most commonly used charts from the other categories. Use the overview charts to get an overall picture of your database activity, then drill down to the more specific reports if you need to. The set includes the following:

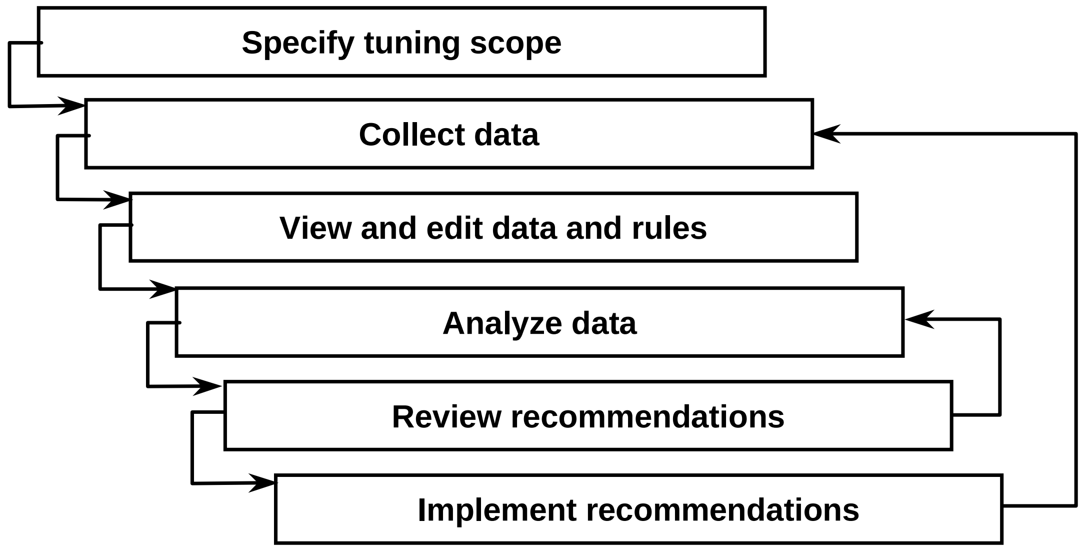
Overview of Throughput is a group chart for the Overview of Performance class. The icon depicts four small bar graphs.

Overview of Performance Default Chart is an individual chart for the Overview of Performance class. The icon depicts a bar graph.

User-Defined Charts

If you have defined any charts of your own, you can select from among them by using this category.

Overview of Oracle Expert Tuning Methodology



Oracle Expert

Oracle Expert is not a part of the basic Oracle Enterprise Manager that is shipped with the database. Rather it is a part of an optional package.

Methodology

Oracle Expert provides automated performance tuning with integrated rules. It automates:

- Collecting data
- Analyzing data
- Generating recommendations
- Creating scripts for recommendations implementation

DBA-Developed Tools

- **Develop your own scripts.**
- **Use the supplied packages for tuning.**
- **Schedule periodic performance checking.**
- **Take advantage of the EM Job service to automate the regular execution of administrative tasks.**
- **Take advantage of the EM Event service to track specific situations.**
- **Take advantage of the EM Job service to apply tasks that automatically solve problems detected by EM event service.**

In-House Scripts

The utilities and Oracle tools may not provide all the statistics you need.

Therefore, you can create your own scripts to do the following:

- Check for free space in every data file
- Determine whether segments have enough space to be able to extend
- Describe schema structures to show tables and associated indexes
- Use Oracle-supplied packages (created by `$ORACLE_HOME/rdbms/admin/dbms*.sql` scripts) (refer to the Oracle Database Administration course for more information)

Summary

In this lesson, you should have learned how to:

- Use the alert log file
- Get information from background processes trace files
- Trace user SQL statements
- Collect statistics from dictionary and dynamic performance troubleshooting views
- Use the `STATSPACK` utility to collect performance data
- Retrieve wait events information

2-35

Practice 2

The goal of this practice is to familiarize yourself with the different methods of collecting statistical information.

1. Log on as directed by the instructor. If the database is not already started, connect to sqlplus using “system/manager as sysdba” then start it using the STARTUP command. Check that TIMED_STATISTICS has been set to true, if it has not then set it using the init.ora file (located at $HOME/ADMIN/PFILE), or the alter system statement.
2. Connect to SQLPLUS as user SYSTEM, and issue a command that will create a trace file for this session. Run a query to count the number of rows in the dictionary view DBA_TABLES. In order to locate your new trace file easier, if possible, delete all the trace files in the USER_DUMP_DEST before running the trace. Disable the trace command after running the query. Do not try to interpret the content of the trace file as this is the topic of a later lesson.
3. At the operating system level view the resulting trace file located in the directory set by USER_DUMP_DEST.
4. Open two session, the first as user HR/HR, and the second as user sys/oracle as sysdba. From the second session generate a user trace file for the first session using DBMS_SYSTEM.SET_SQL_TRACE_IN_SESSION procedure.
5. Confirm that the trace file has been created in the directory set by USER_DUMP_DEST
6. Connect to SQLPLUS using “sys/oracle as sysdba”, and create a new tablespace (TOOLS) to hold the tables and other segments required by STATSPACK. This tablespace needs to be 100M, and be dictionary managed (this will be used later in the course). Name the datafile tools01.dbf and place in the $HOME/ORADATA/u05 directory.

   Note: Dictionary managed is not the default.
7. Confirm, and record, the amount of free space available within the TOOLS tablespace by querying the DBA_FREE_SPACE view.
8. Connect using “sys/oracle as sysdba”, then install STATSPACK using the spcreate.sql script located in your $HOME/STUDENT/LABS directory. Use the following settings when asked by the installation program.
9. Query DBA_FREE_SPACE to determine the amount of free space space left in the TOOLS tablespace. The difference between this value and the one records in question (7) will be the space required for the initial installation of STATSPACK. Note that this amount will increase in proportion to the amount of information stored within the STATSPACK tables, i.e., the number of snapshots.
10. Manually collect current statistics using STATSPACK by running the snap.sql script located in $HOME/STUDENT/LABS. This will return the snap_id for the snapshot just taken, which should be recorded.
11. In order to have STATSPACK automatically collect statistics every 3 minutes execute the spauto.sql script located it your $HOME/STUDENT/LABS directory. Query the database to confirm that the job has been registered using the user_jobs view..

Practice 2 (continued)

12. After waiting for a period in excess of 3 minutes query the stats$snapshot view in order to list what snapshots have been collected. There must be at least two snapshots before moving to the next step.
13. Once there are at least two snapshots, start to generate a report. This is performed using the spreport.sql script found in the $HOME/STUDENT/LABS directory. The script lists the snapshot options available, and then requests the beginning snap id and the end snap id. The user is then requested to give a filename for the report, this is often best left to the default.
14. Locate the report file in the users current directory, then using any text editor, open and examine the report. The first page shows a collection of the most queried statistics.
15. Connect to the database as a system administrator “sys/oracle as sysdba”.
16. Query the database in order to determine what system wait events have been registered since startup using v$system_event.
17. Determine if there are any sessions actually waiting for resources, using v$session_wait.
18. Stop the automatic collection of statistics by removing the job. This is performed by connecting as user perfstat/perfstat and querying the view user_jobs to get the job number. Then execute DBMS_JOB.REMOVE (<job number>);

3

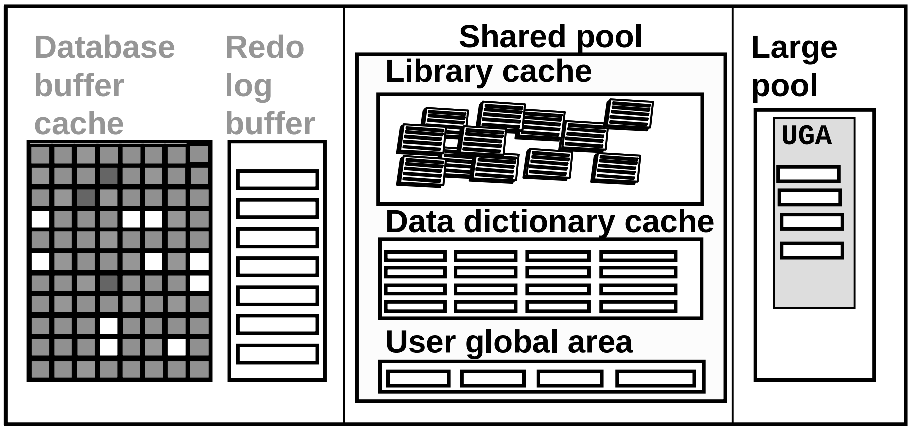
Sizing the Shared Pool

Objectives

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

- Measure and tune the library cache hit ratio
- Measure and tune the dictionary cache hit ratio
- Size and pin objects in the shared pool
- Tune the shared pool reserved space
- Describe the user global area (UGA) and session memory considerations
- List other tuning issues related to the shared pool
- Set the large pool

The System Global Area



Major components of the shared pool are:

- **Library cache**
- **Data dictionary cache**
- **User global area (UGA) for shared server connections**

Shared Pool Contents

The *shared pool* contains the following structures:

- The *library cache,* which stores shared SQL and PL/SQL code
- The *data dictionary cache,* which keeps information about dictionary objects
- The *user global area (UGA),* which keeps information about shared server connections, when Large pool is not configured.

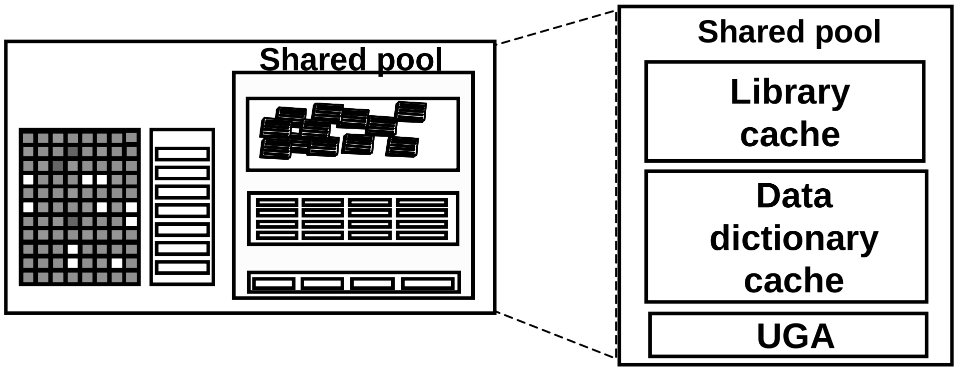
Tuning Considerations for the Shared Pool

A cache miss on the data dictionary cache or library cache is more expensive than a miss on the database buffer cache. Tuning the shared pool is a priority.

When you tune the shared pool, you should concentrate on the library cache, because the algorithm tends to hold dictionary data in memory longer than library cache data. Therefore, tuning the library cache to an acceptable cache hit ratio ensures that the data dictionary cache hit ratio is also acceptable.

If the shared pool is too small, the server must dedicate resources to managing the limited space available. This consumes CPU resources and causes contention.

The Shared Pool



- **Size is defined by `SHARED_POOL_SIZE`.**
- **Library cache contains statement text, parsed code, and execution plan.**
- **Data dictionary cache contains definitions for tables, columns, and privileges from the data dictionary tables.**
- **UGA contains session information for Oracle Shared Server users when large pool is not configured.**

Size of the Shared Pool

You set the size of the shared pool with the SHARED_POOL_SIZE initialization parameter. It defaults to 8,388,608 bytes (8 MB).

The Library Cache

The library cache contains shared SQL and PL/SQL areas: the fully parsed or compiled representations of PL/SQL blocks and SQL statements.

PL/SQL blocks include:

- Procedures
- Functions
- Packages
- Triggers
- Anonymous PL/SQL blocks

The Data Dictionary Cache

The data dictionary cache holds definitions of dictionary objects in memory.

User Global Area

The UGA contains the session information for Oracle Shared server. UGA is located in the Shared pool when using shared server session and if large pool is not configured.

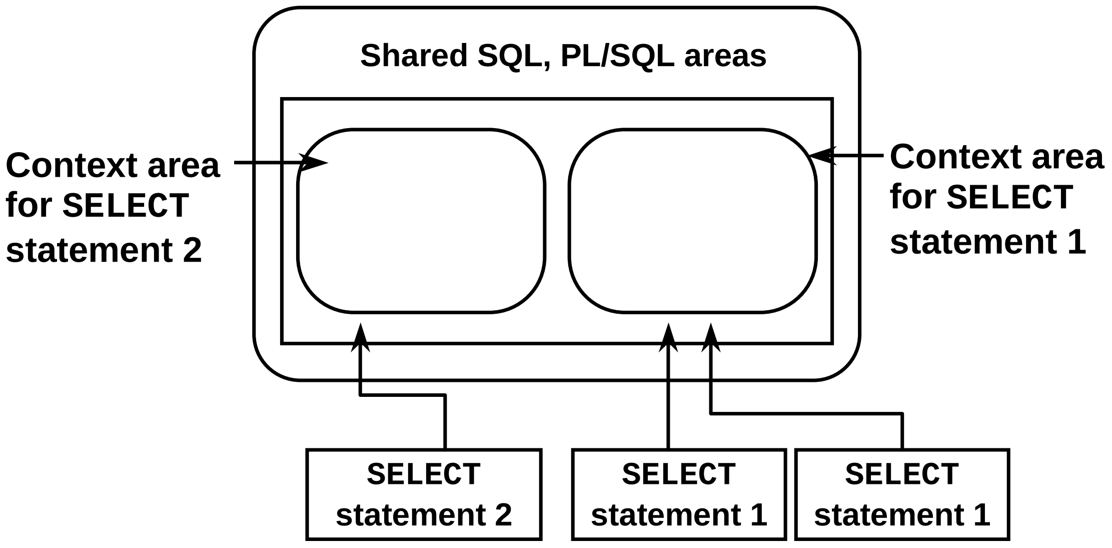
The Library Cache

- Used to store SQL statements and PL/SQL blocks to be shared by users
- Managed by an LRU algorithm
- Used to prevent statements reparsing

ORACLE

3-5

The Library Cache



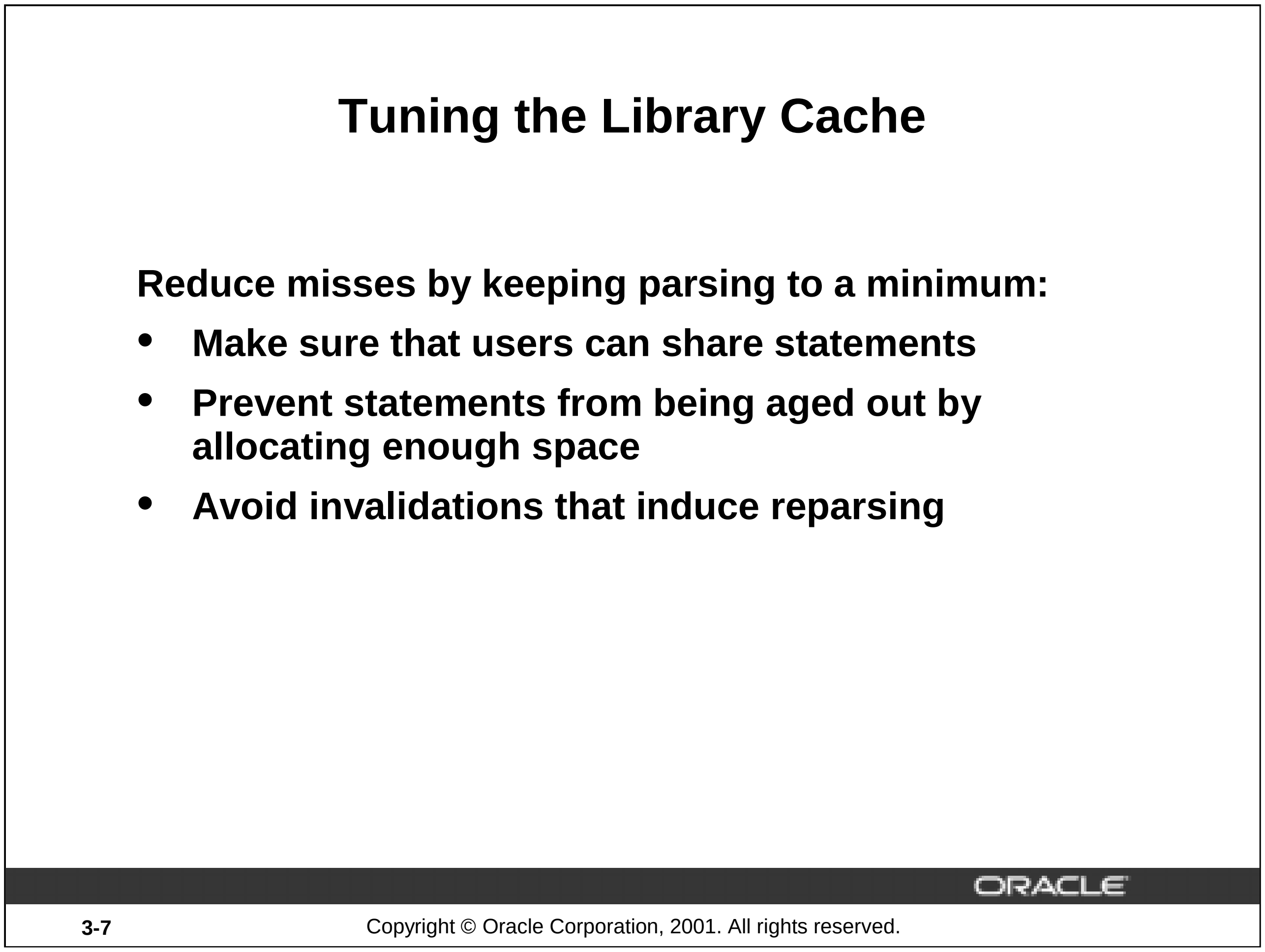
SQL and PL/SQL Storage

The Oracle server uses the library cache to store SQL statements and PL/SQL blocks. A least recently used (LRU) algorithm is used to manage the cache.

If a user issues a statement that is already in the cache, the Oracle server can use the cached version without having to reparse it.

To find whether a statement is already cached, the Oracle server:

- Reduces the statement to the numeric value of the ASCII text
- Uses a hash function of this number



Tuning the Library Cache

Reduce misses by keeping parsing to a minimum:

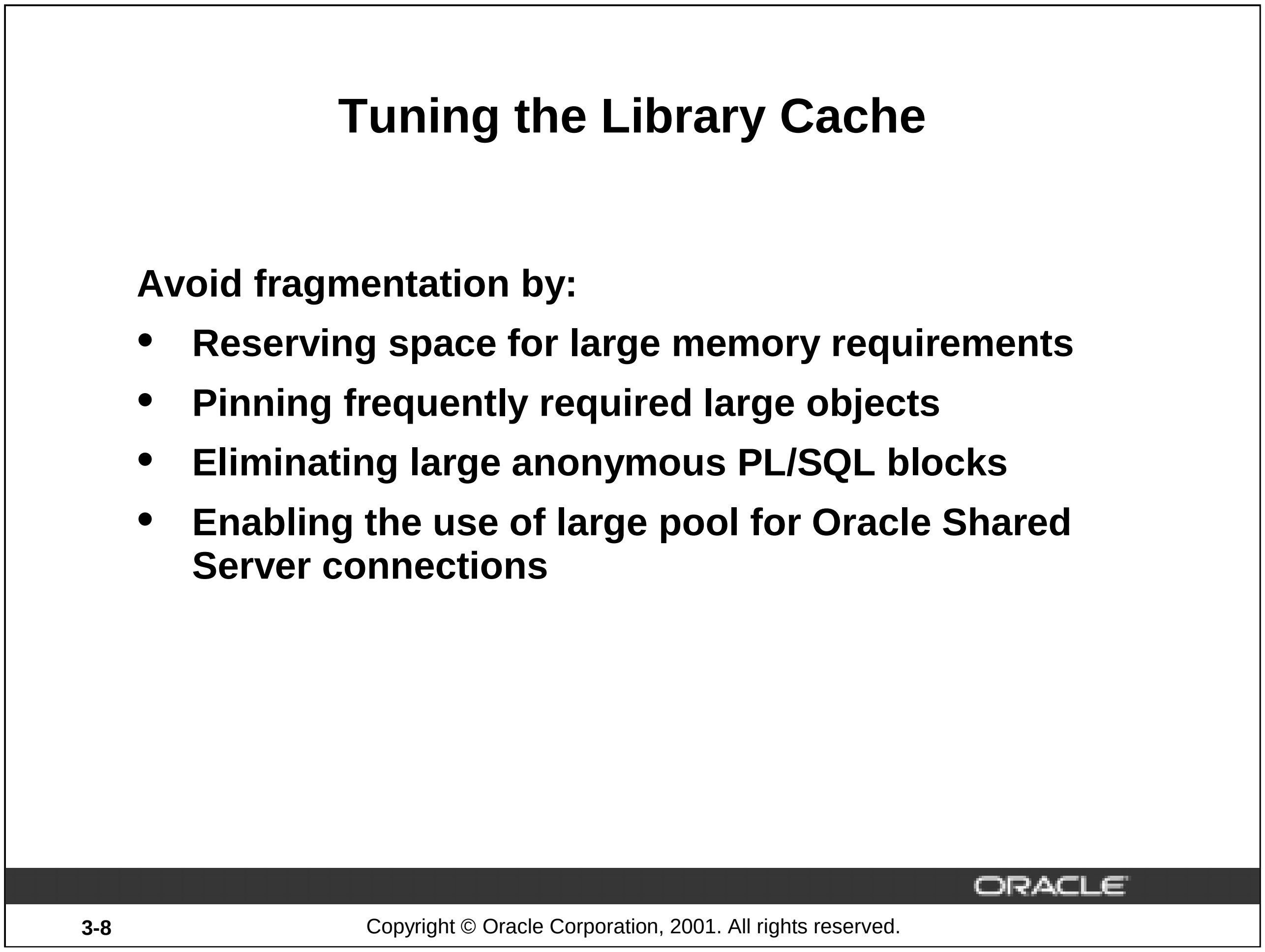
- **Make sure that users can share statements**
- **Prevent statements from being aged out by allocating enough space**
- **Avoid invalidations that induce reparsing**

3-7

Tuning Goal

In an OLTP environment, reparsing should be minimized.

- If an application makes a parse call for a SQL statement and the parsed representation of the statement does not already exist in a shared SQL area in the library cache, the Oracle server parses the statement and allocates a shared SQL area.
- Ensure that SQL statements can share a shared SQL area whenever possible, by using:
  - As much generic code as possible
  - Bind variables rather than constants
- If an application makes an execute call for a SQL statement, and the shared SQL area containing the parsed representation of the statement has been deallocated from the library cache to make room for another statement, the Oracle server implicitly reparses the statement, allocates a new shared SQL area for it, and executes it. Reduce library cache misses on execution calls by allocating more memory to the library cache.
- If a schema object is referenced in a SQL statement and that object is later modified in any way, the SQL statement held in memory is rendered invalid.



Tuning the Library Cache

Avoid fragmentation by:

- **Reserving space for large memory requirements**
- **Pinning frequently required large objects**
- **Eliminating large anonymous PL/SQL blocks**
- **Enabling the use of large pool for Oracle Shared Server connections**

3-8

Second Tuning Goal

Avoid memory fragmentation by:

- Ensuring availability of contiguous space for large memory requirements through the allocation of reserved space in the shared pool area
- Pinning frequently required large objects such as SQL and PL/SQL areas in memory, instead of aging them out with the normal LRU mechanism
- Using small PL/SQL packaged functions instead of large anonymous blocks.
- Configuring the large pool for Oracle Shared Server connections.
- Measuring session memory use by the shared server processes in Oracle Shared Server connections

Terminology

- **Gets: (Parse) The number of lookups for objects of the namespace**
- **Pins: (Execution) The number of reads or executions of the objects of the namespace**
- **Reloads: (Reparse) The number of library cache misses on the execution step, causing implicit reparsing of the statement and block**

Three Keywords

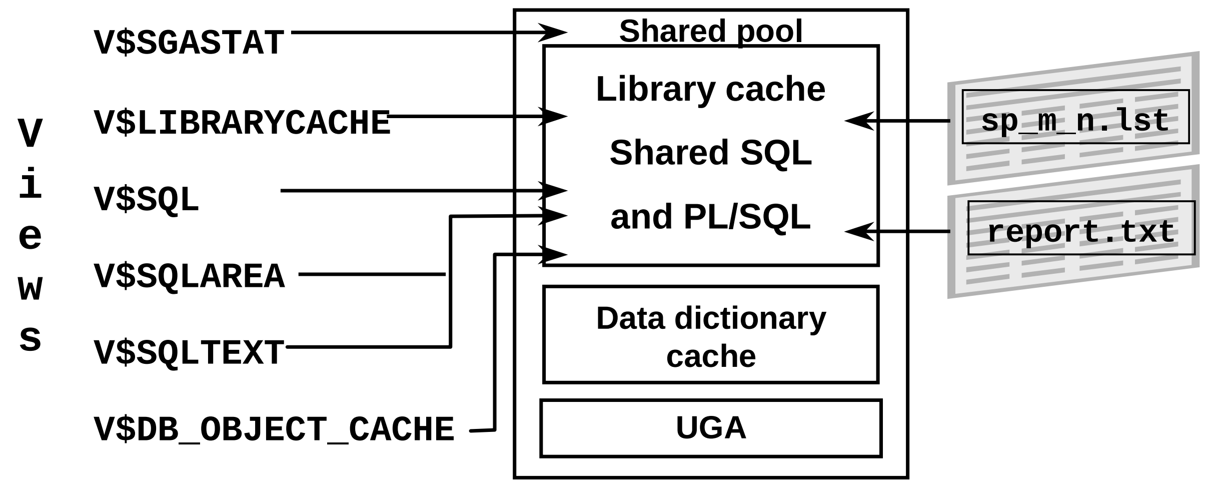
Each row in `V$LIBRARYCACHE` contains statistics for one type of item kept in the library cache. The item described by each row is identified by the value of the `NAMESPACE` column. Rows of the table with the following `NAMESPACE` values reflect library cache activity for SQL statements and PL/SQL blocks: `SQL AREA`, `TABLE/PROCEDURE`, `BODY`, and `TRIGGER`.

Rows with other `NAMESPACE` values reflect library cache activity for object definitions that the server uses for dependency maintenance: `INDEX`, `CLUSTER`, `OBJECT`, and `PIPE`.

Three keywords related to the namespace are:

- `GETS` shows the total number of requests for information on the corresponding item.
- `PINS`: For each of these areas, `PINS` shows the number of executions of SQL statements or procedures.
- `RELOADS`: If an execute call for a SQL statement is performed and the shared SQL area containing the parsed representation of the statement has been deallocated from the library cache to make room for another statement or because the objects the statement refers to have been invalidated, the Oracle server implicitly reloads the statement and therefore reparses it. The number of reloads is counted for each of these namespaces.

Diagnostic Tools for Tuning the Library Cache



Parameters affecting the components
SHARED_POOL_SIZE, OPEN_CURSORS
SESSION_CACHED_CURSORS, CURSOR_SPACE_FOR_TIME

Description of the Views

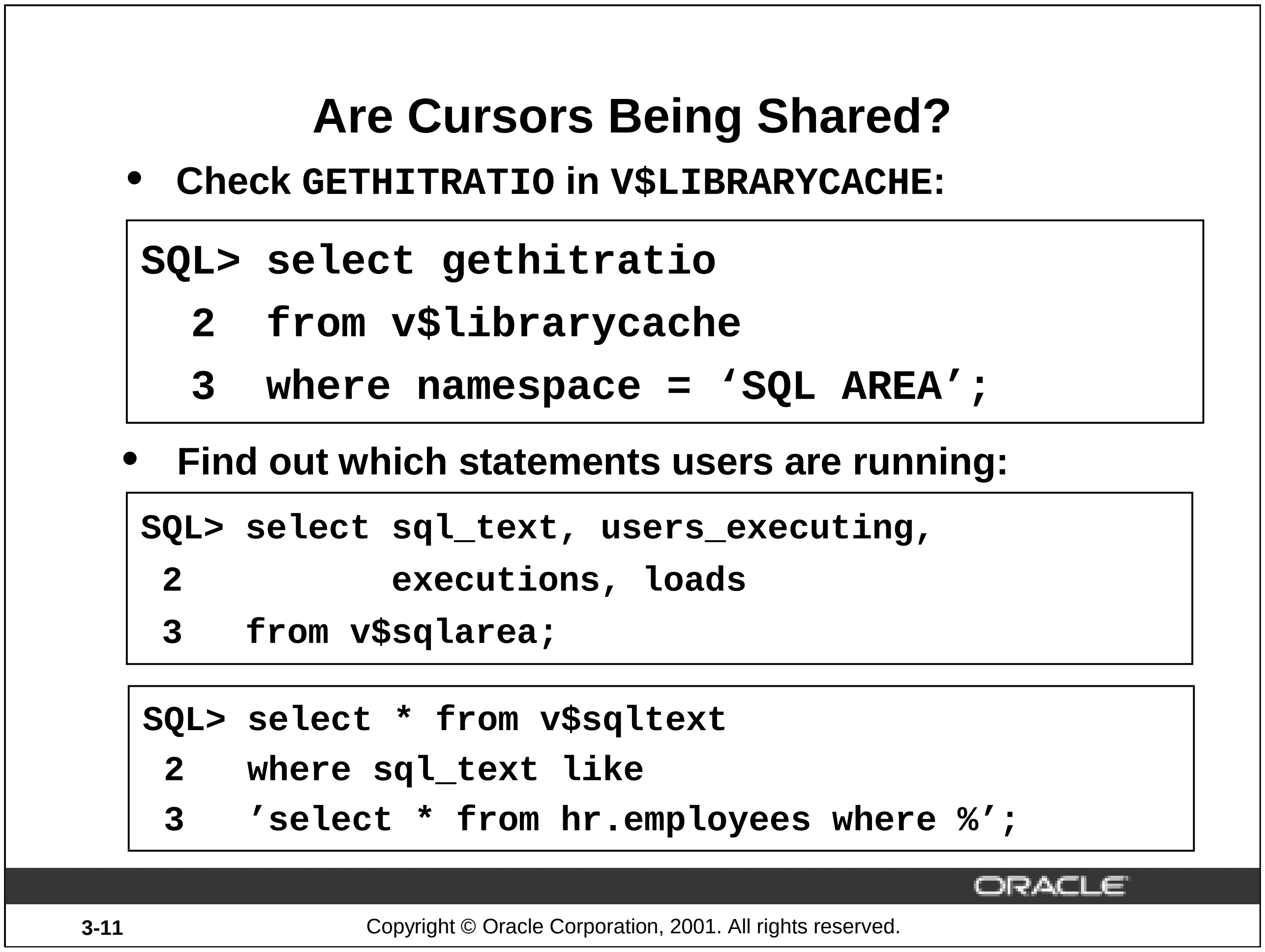
`V$SGASTAT` displays the sizes of all SGA structures. The contents of the shared pool are not aged out as long as free memory is available in the shared pool.

Other dynamic views that help in diagnosing performance issues related to the library cache are:

- `V$LIBRARYCACHE`: Statistics on library cache management
- `V$SQLAREA`: Full statistics about all shared cursors, and the first 1,000 characters of the SQL statement
- `V$SQLTEXT`: The full SQL text without truncation, in multiple rows
- `V$DB_OBJECT_CACHE`: Database objects cached, including packages; also objects such as tables and synonyms, where these are referenced in SQL statements

Example of a STATSPACK Report

```
Check the Instance Efficiency Percentages, Shared Pool
Statistics,  and the Instance Activity Stats related to the
Shared Pool.
```



Are Cursors Being Shared?

- Check GETHITRATIO in V$LIBRARYCACHE:

```
SQL> select gethitratio
  2  from v$librarycache
  3  where namespace = ‘SQL AREA’;
```

- Find out which statements users are running:

```
SQL> select sql_text, users_executing,
 2          executions, loads
 3   from v$sqlarea;
```

```
SQL> select * from v$sqltext
 2   where sql_text like
 3   ’select * from hr.employees where %’;
```

3-11 Copyright © Oracle Corporation, 2001. All rights reserved.

Are Cursors Being Shared?

V$LIBRARYCACHE view: GETHITRATIO determines the percentage of parse calls that find a cursor to share (GETHITS/GETS). This ratio should be in the high 90s in OLTP environments. If not, you can improve the efficiency of your application code.

```
SQL> select namespace,gethitratio
  2  from v$librarycache;
NAMESPACE           GETHITRATIO
---------------    -----------
SQL AREA           .86928702
TABLE/PROCEDURE    .80073801
BODY               .6
```

Guidelines: Library Cache Reloads

Executes `PROC1` —> 1st pin, 1 load
Executes `PROC1` —> 2nd pin, no reload
Executes `PROC1` —> 3rd pin, no reload
Executes `PROC1` —> 4th pin, no reload

→ 4 pins and no reloads

- **Reloads should be less than 1% of the pins:**

```
SQL> select sum(pins) "Executions", sum(reloads)
  2         "Cache Misses", sum(reloads)/sum(pins)
  3  from v$librarycache;
```

- **If the reloads-to-pins ratio is greater than 1%, increase the value of the `SHARED_POOL_SIZE` parameter.**

ORACLE

How to Get the Reloads-to-Pins Ratio

- `V$LIBRARYCACHE` view: The view displays whether statements that have already been parsed have been aged out of the cache. The number of reloads should not be more than 1% of the number of pins.
- `STATSPACK` report: Library Cache Activity section contains the summarized information for the instance. The Instance Activity Stats section contains the detailed statistics of the activities. Some of the statistics to look for include – opened cursors cumulative, parse count (failures), parse count (hard), parse count (total).

```
Instance Activity Stats for DB: ED31Instance: ed31Snaps: 1-2
Statistic                    Total   per Second    per Trans
--------------------- ---------- ------------ -----------
parse count (failures)     2          0.0           2.0
parse count (hard)        26          0.0          26.0
parse count (total)      690          0.6         690.0
```

- `report.txt` output: This section indicates the same ratio for the period when `utlbstat` and `utlestat` ran.

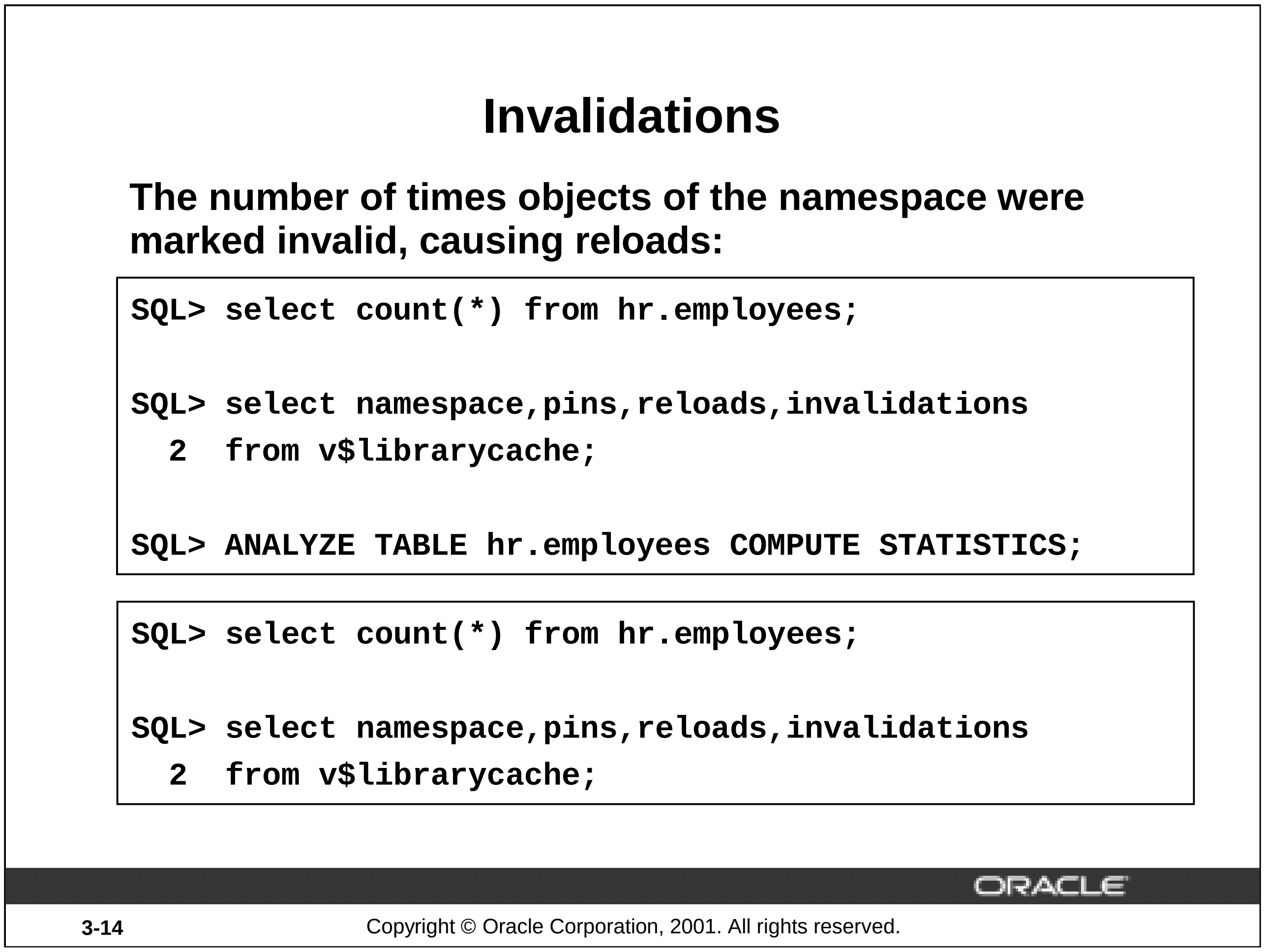
How to Get the Reloads-to-Pins Ratio (continued)

Guidelines

There are two possible reasons for the reloads-to-pins ratio being greater than 1%:

- Though required by successive reexecutions, shared parsed areas have been aged out because of lack of space.
- Shared parsed areas are invalidated.

To avoid these frequent reloads, increase the `SHARED_POOL_SIZE` `init.ora` parameter.



Invalidations

The number of times objects of the namespace were marked invalid, causing reloads:

```
SQL> select count(*) from hr.employees;

SQL> select namespace,pins,reloads,invalidations
  2  from v$librarycache;

SQL> ANALYZE TABLE hr.employees COMPUTE STATISTICS;
```

```
SQL> select count(*) from hr.employees;

SQL> select namespace,pins,reloads,invalidations
  2  from v$librarycache;
```

ORACLE

3-14

When Invalidations Occur

If a schema object is referenced in a SQL statement and that object is later modified in any way, the shared SQL area becomes invalidated (marked as invalid), and the statement must be reparsed the next time it is executed, and therefore reloaded.

For example, when a table, sequence, synonym, or view is re-created or altered or dropped, or a procedure or package specification is recompiled, all dependent shared SQL areas are invalidated.

Example

```
SQL> select count(*) from hr.employees;

SQL> select namespace,pins,reloads,invalidations from
v$librarycache;

NAMESPACE             PINS    RELOADS INVALIDATIONS

--------------------- ---------- ---------- ------------

SQL AREA              1616        12            0

…

SQL> analyze table hr.employees compute statistics;

Table analyzed.
```

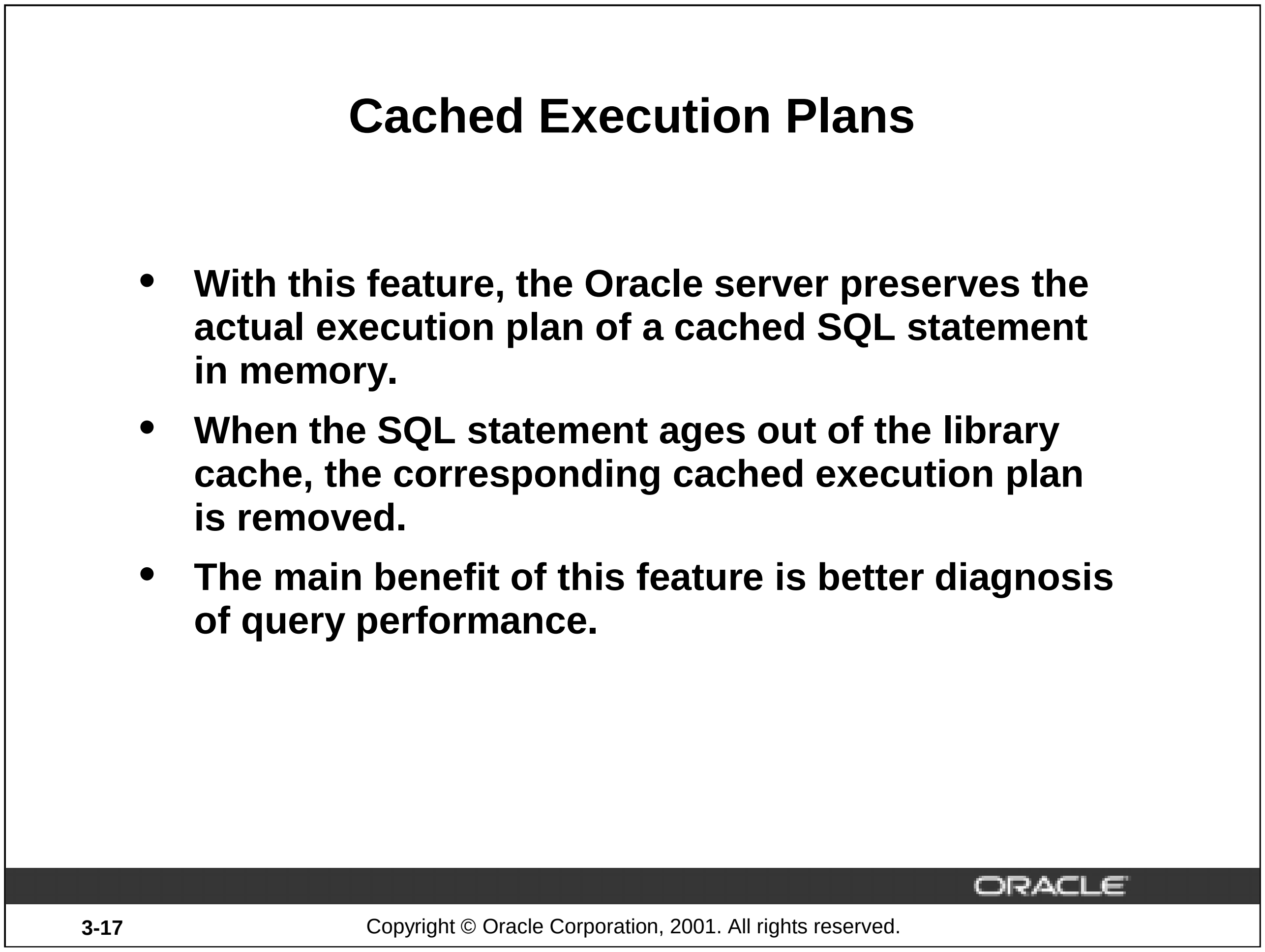
When Invalidations Occur (continued)

```
SQL> select count(*) from hr.employees;

SQL> select namespace,pins,reloads,invalidations from
v$librarycache;

NAMESPACE             PINS    RELOADS INVALIDATIONS

--------------- ---------- ---------- -------------

SQL AREA              1688         14             3

…
```

Sizing the Library Cache

- Define the global space necessary for stored objects (packages, views, and so on).
- Define the amount of memory used by the usual SQL statements.
- Reserve space for large memory requirements in order to avoid misses and fragmentation.
- Keep frequently used objects.
- Convert large anonymous PL/SQL blocks into small anonymous blocks calling packaged functions.

3-16



Cached Execution Plans

- **With this feature, the Oracle server preserves the actual execution plan of a cached SQL statement in memory.**
- **When the SQL statement ages out of the library cache, the corresponding cached execution plan is removed.**
- **The main benefit of this feature is better diagnosis of query performance.**

3-17

Cached Execution Plans

Introduced in Oracle9*i*, this feature enables the Oracle server to retain the execution plans in memory as long as the SQL statement remains in library cache. In pre-Oracle9*i* versions, the execution plan was not retained after the statement was compiled.

View to Support Cached Execution Plans

- **A dynamic performance view, `V$SQL_PLAN`, can be used to view the actual execution plan information for cached cursors.**
- **The view contains all of the `PLAN_TABLE` columns (with the exception of the `LEVEL` column), in addition to seven new columns.**
- **Columns that are also present in the `PLAN_TABLE` have the same value as those in `V$SQL_PLAN`.**
- **A new column is added to `V$SQL` .**

ORACLE

New View to Support Cached Execution Plan

The `V$SQL_PLAN` view displays the actual execution plan information for a cached cursor. The view contains all of the `PLAN_TABLE` columns (with the exception of the `LEVEL` column) and seven new columns. The columns present in the `PLAN_TABLE` have the same values as those in the `V$SQL_PLAN`.

Additional `V$SQL_PLAN` columns not found in `PLAN_TABLE:`

- `ADDRESS`: Cursor parent handle address
- `HASH_VALUE`: Parent statement hash value in library cache
- `CHILD_NUMBER`: Number using this execution plan
- `DEPTH`: Level of the operation in the tree
- `CPU_COST`: CPU cost of the operation as estimated by the cost-based optimizer. If using the rule-based optimizer, this column is null.
- `IO_COST`: Cost of the operation as estimated by the cost-based optimizer. If using the rule-based optimizer, this column is Null.
- `TEMP_SPACE`: Space usage of sort or hash-join estimated by cost-based optimizer

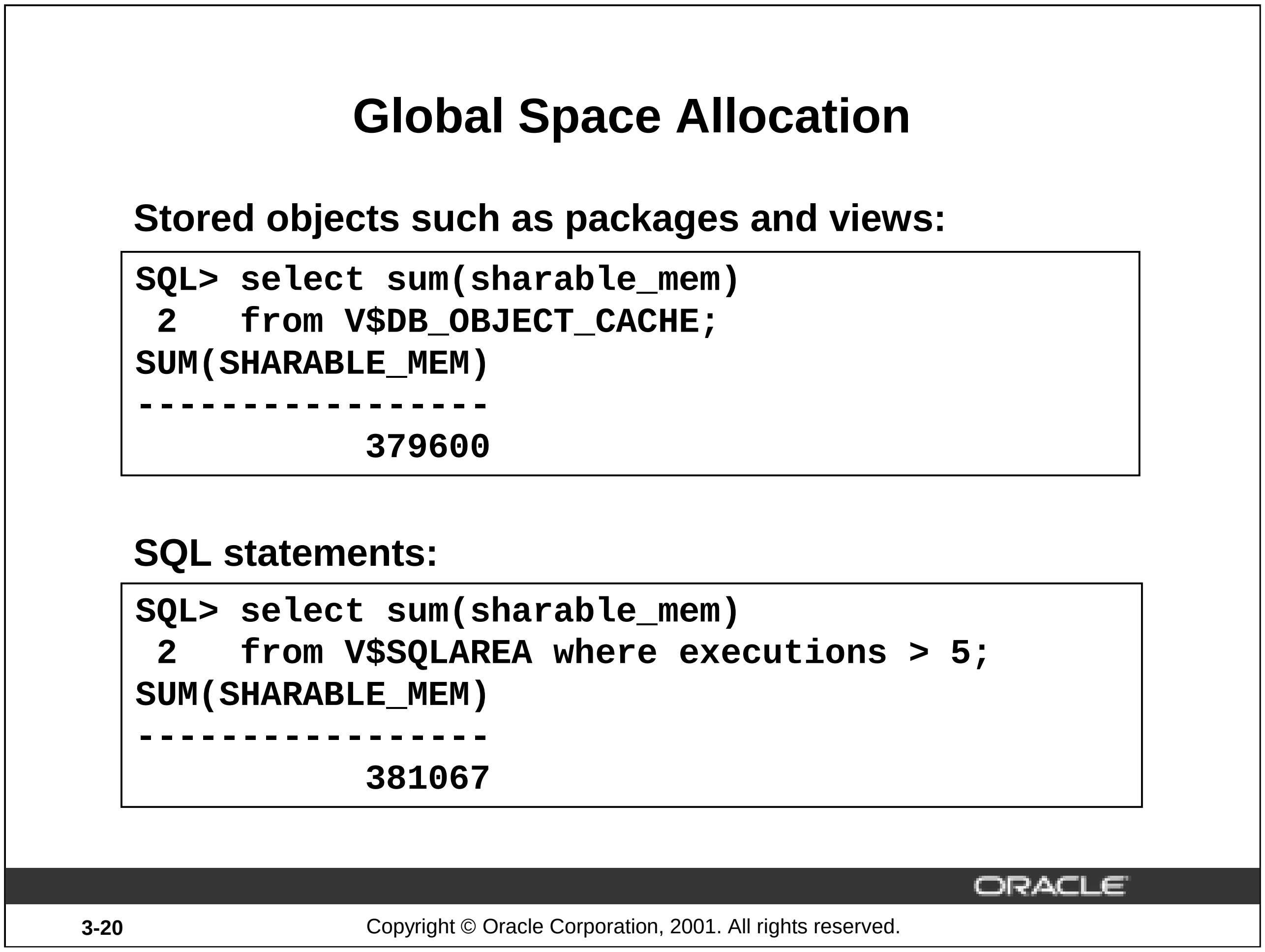
New Column Added to V$SQL to Support Cached Execution Plans

- A new column, PLAN_HASH_VALUE, has been added to the V$SQL view.
- The column information is a hash value built from the corresponding execution plan.
- The column can be used to compare cursor plans the same way the HASH_VALUE column is used to compare cursor SQL texts.

New View to Support Cached Execution Plan (continued)

A new column, PLAN_HASH_VALUE, has been added to the V$SQL view.The column information is a hash value built from the corresponding execution plan.

The column can be used to compare cursor plans the same way the HASH_VALUE column is used to compare cursor SQL texts.



Global Space Allocation

Stored objects such as packages and views:

```
SQL> select sum(sharable_mem)
 2   from V$DB_OBJECT_CACHE;
SUM(SHARABLE_MEM)
-----------------
           379600
```

SQL statements:

```
SQL> select sum(sharable_mem)
 2   from V$SQLAREA where executions > 5;
SUM(SHARABLE_MEM)
-----------------
           381067
```

3-20

Testing Your Applications

For an existing application, you can set up a test and use the dynamic views to find out how much memory is used. Begin by setting `SHARED_POOL_SIZE` to a very large value (at the expense of other structures, if necessary), then run the application.

Computation of Shareable Memory Used

For stored objects such as packages and views, use the following query:

```
SQL>  SELECT SUM(sharable_mem)
2     FROM v$db_object_cache
3     WHERE type = ’PACKAGE’ or type = ’PACKAGE BODY’
4      OR type = ’FUNCTION’ or type = ’PROCEDURE’;
```

For SQL statements, you need to query `V$SQLAREA` after the application has been running for a while. For frequently issued statements, you can use the following query to estimate the amount of memory being used, though this will not include dynamic SQL:

```
SQL> SELECT SUM(sharable_mem)
2       FROM v$sqlarea
3       WHERE executions > 5;
```

Computation of Shareable Memory Used (continued)

You should also allow about 250 bytes in the shared pool per user per open cursor. This can be tested during peak times with the following query:

```
SQL> SELECT SUM(250 * users_opening)
2       FROM v$sqlarea;
```

In a test environment, you can measure shareable memory by selecting the number of open cursors for a test user. You multiply the resulting value by the total number of users:

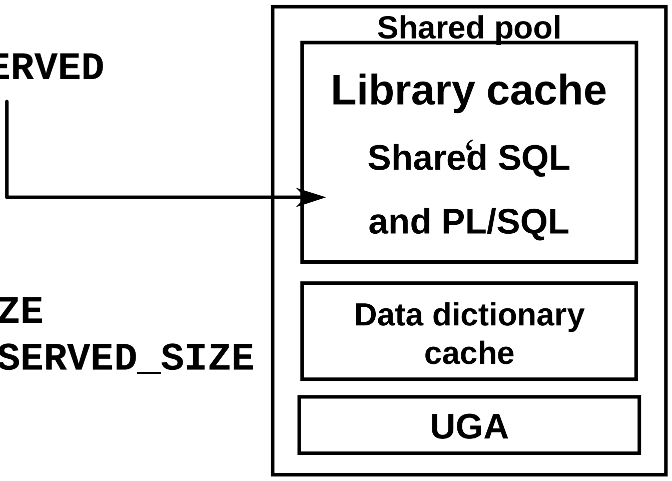
```
SQL> SELECT 250 * value bytes_per_user
2       FROM v$sesstat s, v$statname n
3       WHERE s.statistic# = n.statistic#
4       AND n.name = ’opened cursors current’
5       AND s.sid = 15;
```

Ideally, your application should have a library cache as large as the sum of the numbers above, plus a small allowance for dynamic SQL.

Large Memory Requirements

- **Satisfy requests for large contiguous memory**
- **Reserve contiguous memory within the shared pool**

`V$SHARED_POOL_RESERVED`



`SHARED_POOL_SIZE`
`SHARED_POOL_RESERVED_SIZE`

Why Reserve Space in the Shared Pool?

The DBA can reserve memory within the shared pool to satisfy large allocations during operations such as PL/SQL compilation and trigger compilation. Smaller objects will not fragment the reserved list, helping to ensure that the reserved list has large contiguous chunks of memory. Once the memory allocated from the reserved list is freed, it returns to the reserved list.

Initialization Parameter

The size of the reserved list, as well as the minimum size of the objects that can be allocated from the reserved list, are controlled by the `SHARED_POOL_RESERVED_SIZE` initialization parameter, which controls the amount of `SHARED_POOL_SIZE` reserved for large allocations. Set the initial value to 10% of the `SHARED_POOL_SIZE`. If `SHARED_POOL_RESERVED_SIZE` is greater that half of `SHARED_POOL_SIZE`, the server signals an error.

`V$SHARED_POOL_RESERVED` View

This view helps in tuning the reserved pool and space within the shared pool.

The columns of the view are only valid if the `SHARED_POOL_RESERVED_SIZE` parameter is set to a valid value:

```
SQL> desc V$SHARED_POOL_RESERVED
 Name      Null?            Type
 --------------------       --------      --------
 FREE_SPACE                               NUMBER
 AVG_FREE_SIZE                            NUMBER
 FREE_COUNT                               NUMBER
 MAX_FREE_SIZE                            NUMBER
 USED_SPACE                               NUMBER
 AVG_USED_SIZE                            NUMBER
 USED_COUNT                               NUMBER
 MAX_USED_SIZE                            NUMBER
 REQUESTS                                 NUMBER
 REQUEST_MISSES                           NUMBER
 LAST_MISS_SIZE                           NUMBER
 MAX_MISS_SIZE                            NUMBER
```

where:	*FREE_SPACE*	is the total free space in the reserved list
	AVG_FREE_SIZE	is the average size of the free memory on the reserved list
	MAX_FREE_SIZE	is the size of the largest free piece of memory on the reserved list
	REQUEST_MISSES	is the number of times the served list did not have a free piece of memory to satisfy the request, and proceeded to start flushing objects from the LRU list

The following columns in the view contain values that are valid even if the parameter is not set:

- `REQUEST_FAILURES`
- `LAST_FAILURE_SIZE`
- `ABORTED_REQUEST_THRESHOLD`
- `ABORTED_REQUESTS`
- `LAST_ABORTED_SIZE`

where:	*REQUEST_FAILURES*	is the number of times that no memory was found to satisfy a request
	LAST_FAILURE_SIZE	is the size of the last failed request

Tuning the Shared Pool Reserved Space

- **Diagnostic tools for tuning:**
  - **The `V$SHARED_POOL _RESERVED` dictionary view**
  - **The supplied package and procedure:**
    - **`DBMS_SHARED_POOL`**
    - **`ABORTED_REQUEST_THRESHOLD`**
- **Guidelines: Set the parameter `SHARED_POOL_RESERVED_SIZE`**

ORACLE

3-24 Copyright © Oracle Corporation, 2001. All rights reserved.

Diagnostics with the V$SHARED_POOL_RESERVED View

Statistics from the V$SHARED_POOL_RESERVED view can help you tune the parameters. On a system with ample free memory to increase the SGA, the goal is to have REQUEST_MISSES equal 0, not to have any request failures, or at least to prevent this value from increasing.

Diagnostics with ABORTED_REQUEST_THRESHOLD Procedure

The ABORTED_REQUEST_THRESHOLD procedure, in the DBMS_SHARED_POOL package, enables you to limit the amount of the shared pool to flush prior to reporting an ORA-4031 error, so as to limit the extent of a flush that could occur due to a large object.

Guidelines When SHARED_POOL_RESERVED_SIZE Is Too Small

The reserved pool is too small when the value for REQUEST_FAILURES is more than zero and increasing. To resolve this, you can increase the value of SHARED_POOL_RESERVED_SIZE and SHARED_POOL_SIZE accordingly. The settings you select for these depend on your system's SGA size constraints.

This option increases the amount of memory available on the reserved list without having an effect on users who do not allocate memory from the reserved list. As a second option, reduce the number of allocations allowed to use memory from the reserved list; doing so, however, increases the normal shared pool, which may have an effect on other users on the system.

Guidelines When SHARED_POOL_RESERVED_SIZE Is Too Large

Too much memory may have been allocated to the reserved list if:

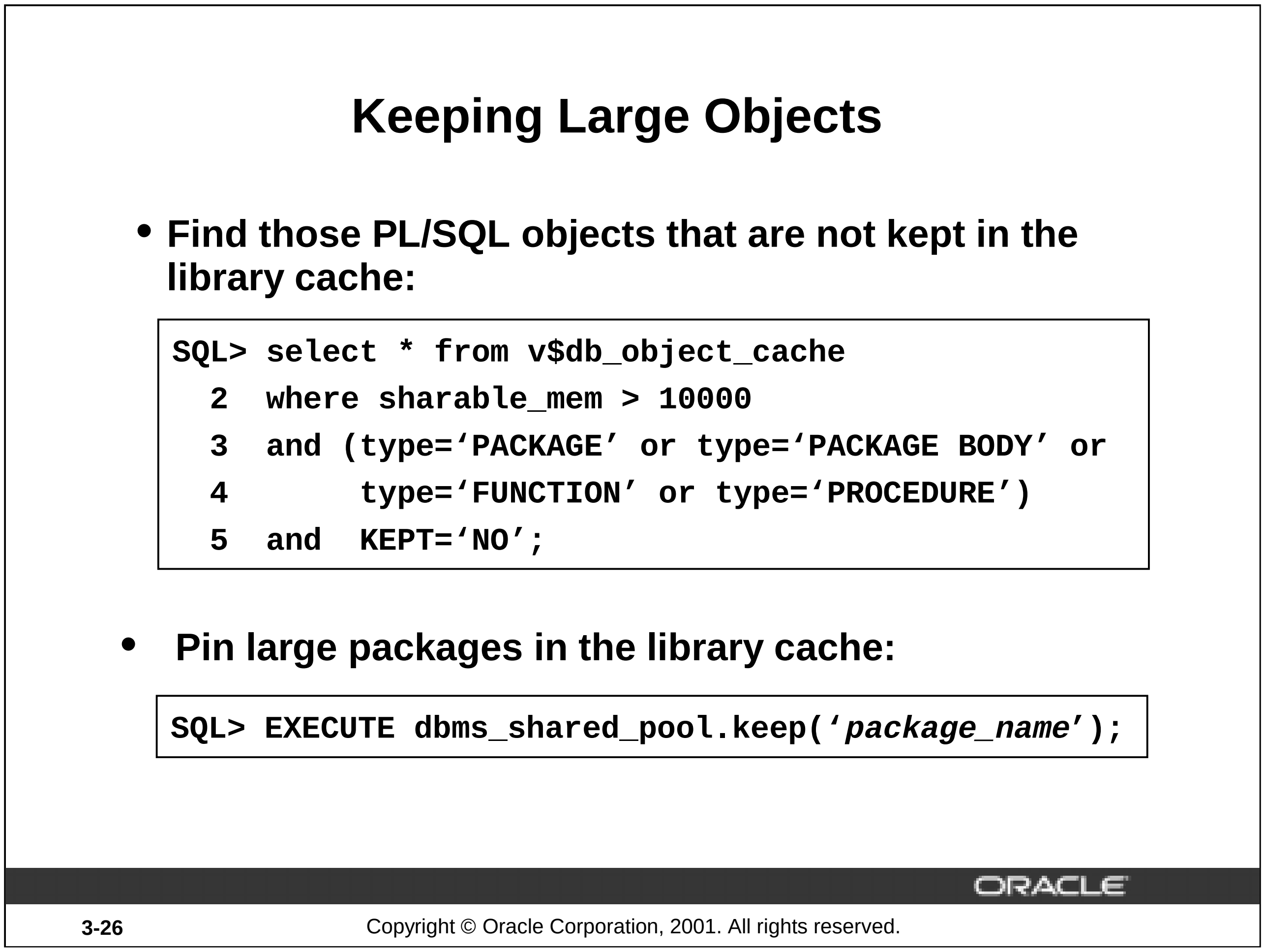
- REQUEST_MISS = 0 or not increasing
- FREE_MEMORY = > 50% of the SHARED_POOL_RESERVED_SIZE minimum

If either of these is true, decrease the value for SHARED_POOL_RESERVED_SIZE.

Guidelines When SHARED_POOL_SIZE Is Too Small

The V$SHARED_POOL_RESERVED fixed table can also indicate when the value for SHARED_POOL_SIZE is too small. This may be the case if REQUEST_FAILURES > 0 and increasing.

Then, if you have enabled the reserved list, decrease the value for SHARED_POOL_RESERVED_SIZE. If you have not enabled the reserved list, you could increase SHARED_POOL_SIZE.



Keeping Large Objects

- **Find those PL/SQL objects that are not kept in the library cache:**

```
SQL> select * from v$db_object_cache
  2  where sharable_mem > 10000
  3  and (type='PACKAGE' or type='PACKAGE BODY' or
  4       type='FUNCTION' or type='PROCEDURE')
  5  and  KEPT='NO';
```

- **Pin large packages in the library cache:**

```
SQL> EXECUTE dbms_shared_pool.keep('package_name');
```

3-26

Why and When to Keep Objects

Loading large objects is the primary source of fragmentation. Users' response time is affected because of the large number of small objects that need to be aged out from the shared pool to make room. To prevent these situations, keep these large or frequently-required objects in the shared pool to make sure that they are never aged out of the shared pool.

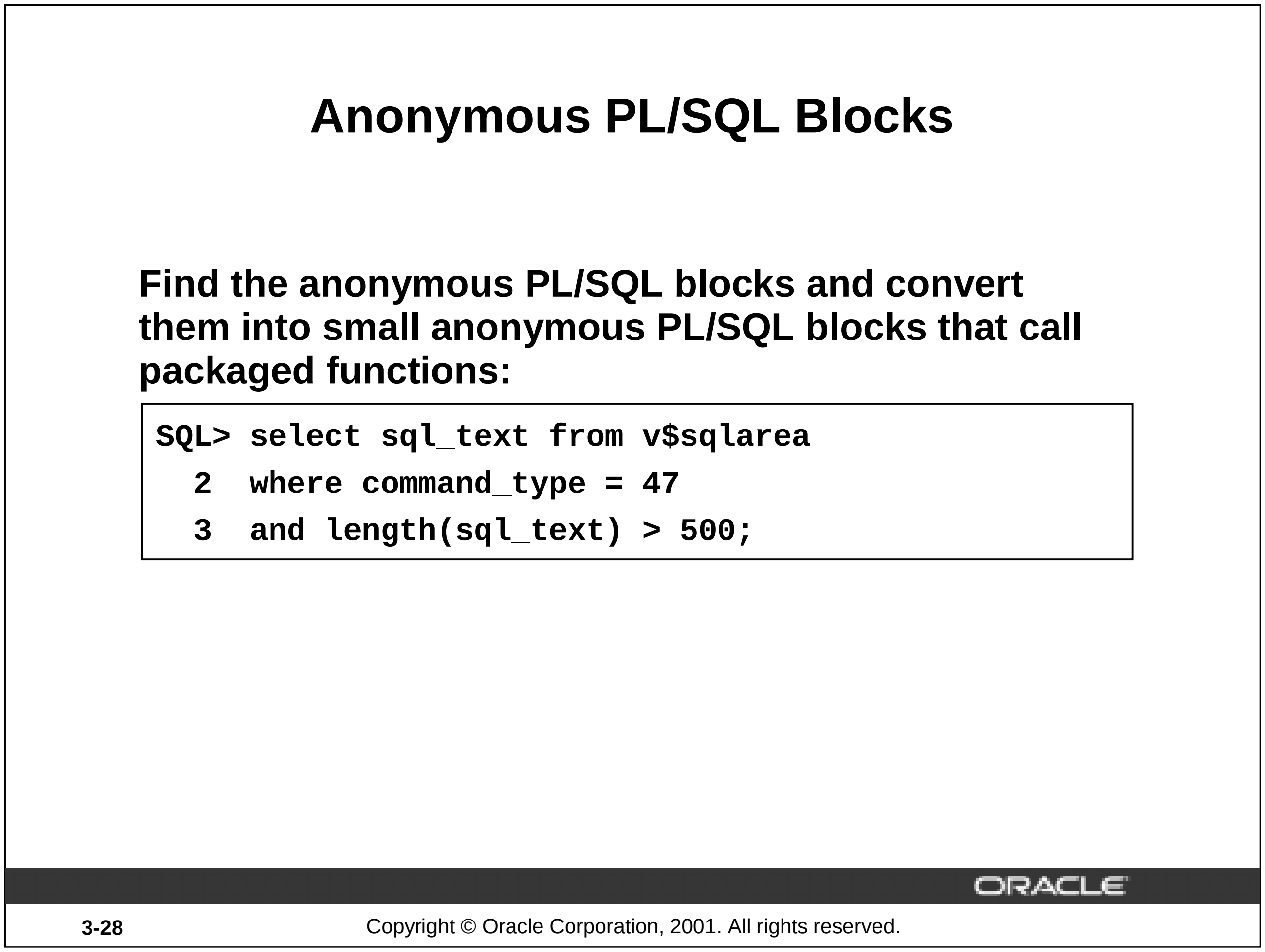
- Which objects to keep:
  - Frequently-required large procedural objects such as `STANDARD`, `DIUTIL` packages, and those for which shareable memory exceeds a defined threshold
  - Compiled triggers that are executed often on frequently used tables
  - Sequences, because sequence numbers are lost when the sequence is aged out of the shared pool
- When to keep them: Startup time is best, because that prevents further fragmentation.
- Flushing the shared pool using the command `ALTER SYSTEM FLUSH SHARED_POOL` does not flush kept objects.

How to Keep Objects

Use the supplied `DBMS_SHARED_POOL` package and the `KEEP` procedure to keep objects.

To create the package, run the `dbmspool.sql` script. The `prvtpool.plb` script is automatically executed at the end of the previous one. These scripts are not run by `catproc.sql.`

Use the `UNKEEP` procedure to remove pinned objects from the shared pool.



Anonymous PL/SQL Blocks

Find the anonymous PL/SQL blocks and convert them into small anonymous PL/SQL blocks that call packaged functions:

```
SQL> select sql_text from v$sqlarea
  2  where command_type = 47
  3  and length(sql_text) > 500;
```

Eliminating Large Anonymous PL/SQL Blocks

Two Solutions

- Find them and convert them into small anonymous PL/SQL blocks that call packaged functions.
- If an anonymous PL/SQL block cannot be turned into a package, it can be identified in `V$SQLAREA` and marked KEPT.

You can then keep these blocks in memory, using the appropriate supplied procedure.

```
SQL> declare x number;
 2>   begin x := 5;
 3>   end;
```

Should be written as :

```
SQL>  declare /* KEEP_ME */ x number;
 2>   begin x := 5;
 3>   end;
```

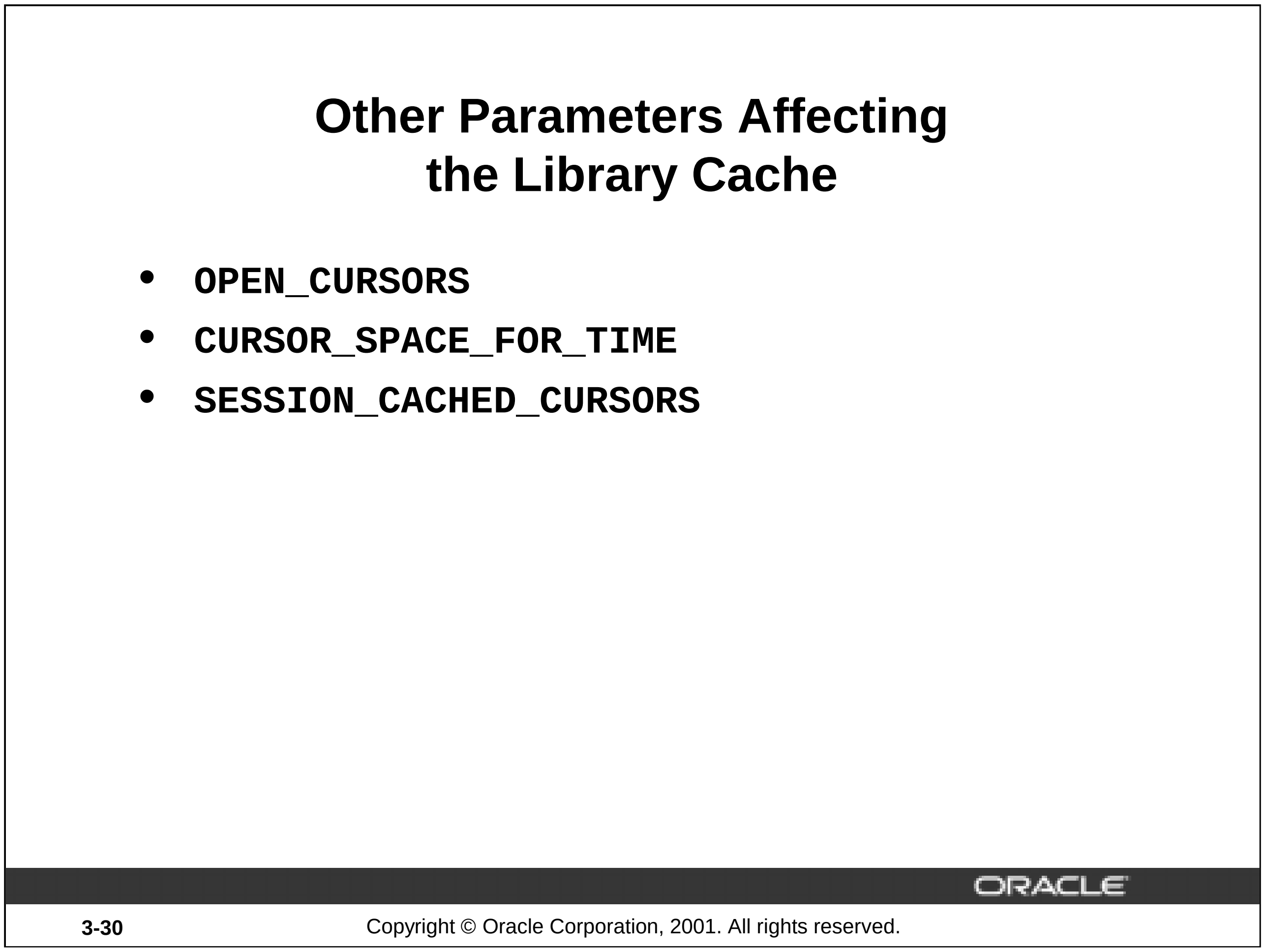
You can locate this statement with the following query:

```
SQL>  select address, hash_value
  2>  from v$sqlarea
  3>  where command_type = 47 and sql_text like ’%KEEP_ME%’;
```

Eliminating Large Anonymous PL/SQL Blocks (continued)

Then execute the `KEEP` procedure on the anonymous PL/SQL block identified by the address and hash value retrieved from the previous statement:

```
SQL>  execute dbms_shared_pool.keep('address,hash_value');
```



Other Parameters Affecting the Library Cache

- **`OPEN_CURSORS`**
- **`CURSOR_SPACE_FOR_TIME`**
- **`SESSION_CACHED_CURSORS`**

3-30

Initialization Parameters

The following parameters also affect the library cache:

- `OPEN_CURSORS`: This parameter defines the number of cursors referencing private SQL areas allocated to the user's process. A private SQL area continues to exist until the cursor is closed and therefore still exists after the completion of the statement.

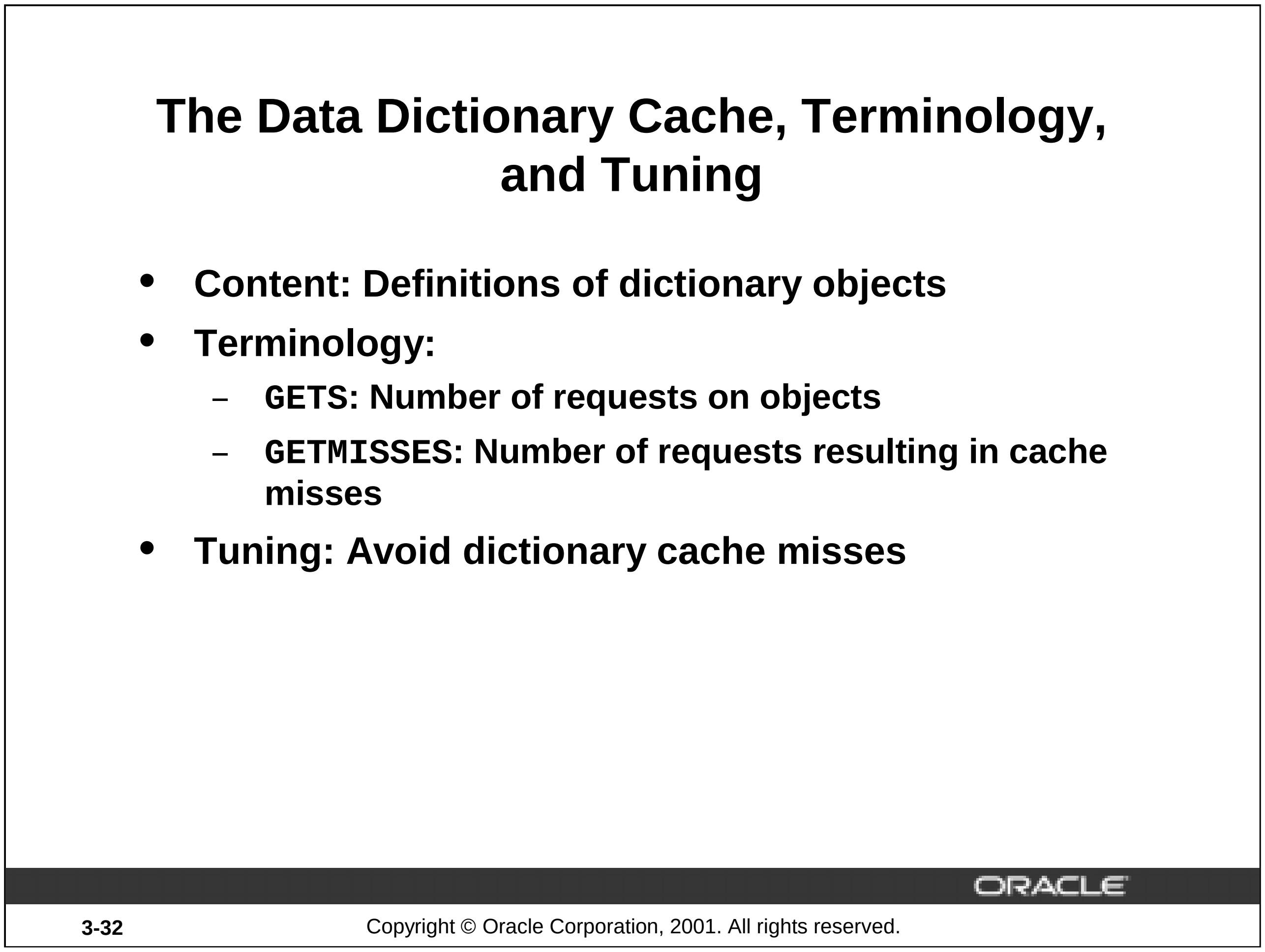
  To take advantage of additional memory available for shared SQL areas, you may need to increase the number of cursors permitted for a session. Application developers should close unneeded open cursors to conserve system memory. The default value is 50.
- `CURSOR_SPACE_FOR_TIME`: This is a boolean parameter that defaults to FALSE. If you set it to TRUE, you choose to use space to gain time; shared SQL areas are not aged out until the cursor referencing them is closed. Therefore, make sure that there is free memory and no cache misses. Do not change this parameter unless the value of `RELOADS` in `V$LIBRARYCACHE` is consistently 0. If your application uses Forms, or any dynamic SQL, leave the setting at FALSE.

Initialization Parameters (continued)

- `SESSION_CACHED_CURSORS`: This parameter helps in situations in which a user repeatedly parses the same statements. This occurs in Forms applications when users often switch between forms; all the SQL statements opened for a form are closed when you switch to another one. The parameter causes closed cursors to be cached within the session. Therefore, any subsequent call to parse the statement will bypass the parse phase. (This is similar to `HOLD_CURSORS` in the precompilers.)

  To check that your setting is efficient, compare the “session cursor cache hits” and “parse count” session statistics in `V$SESSTAT` for a typical user session. If few parses result in hits, you might increase the number. Remember that this increases overall demands on memory.

  The default is 0, which means no caching.



The Data Dictionary Cache, Terminology, and Tuning

- **Content: Definitions of dictionary objects**
- **Terminology:**
  - **`GETS`: Number of requests on objects**
  - **`GETMISSES`: Number of requests resulting in cache misses**
- **Tuning: Avoid dictionary cache misses**

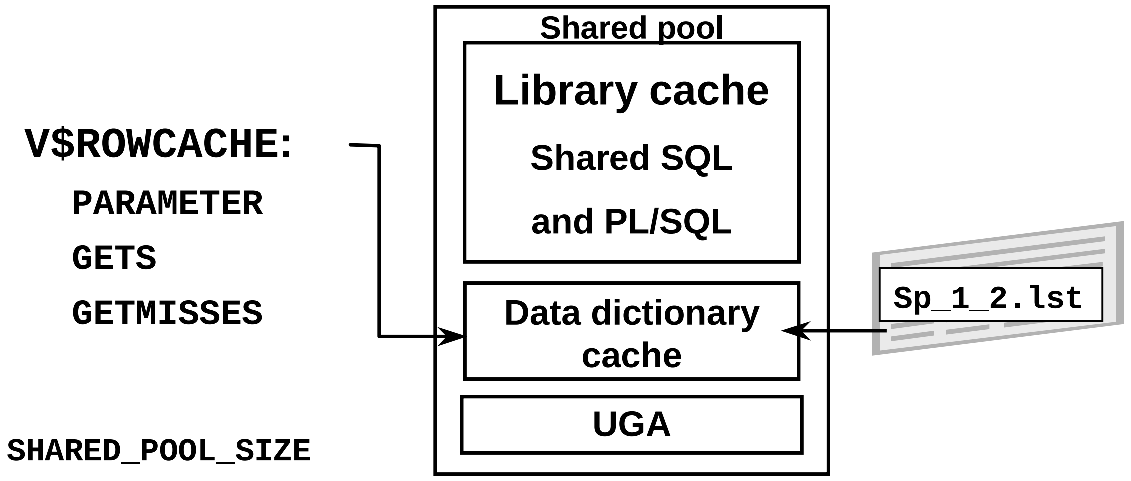
Two Keywords

- `GETS`: Shows the total number of requests for information on the corresponding item (for example, in the row that contains statistics for file descriptions, this column has the total number of requests for file description data)
- `GETMISSES`: Shows the number of data requests resulting in cache misses

Goal

Misses on the data dictionary cache are to be expected in some cases. Upon instance startup, the data dictionary cache contains no data, so any SQL statement issued is likely to result in cache misses. As more data is read into the cache, the likelihood of cache misses should decrease. Eventually, the database should reach a “steady state” in which the most frequently used dictionary data is in the cache. At this point, very few cache misses should occur. To tune the cache, examine its activity only after your application has been running.

Diagnostic Tools for Tuning the Data Dictionary Cache



3-33 Copyright © Oracle Corporation, 2001. All rights reserved.

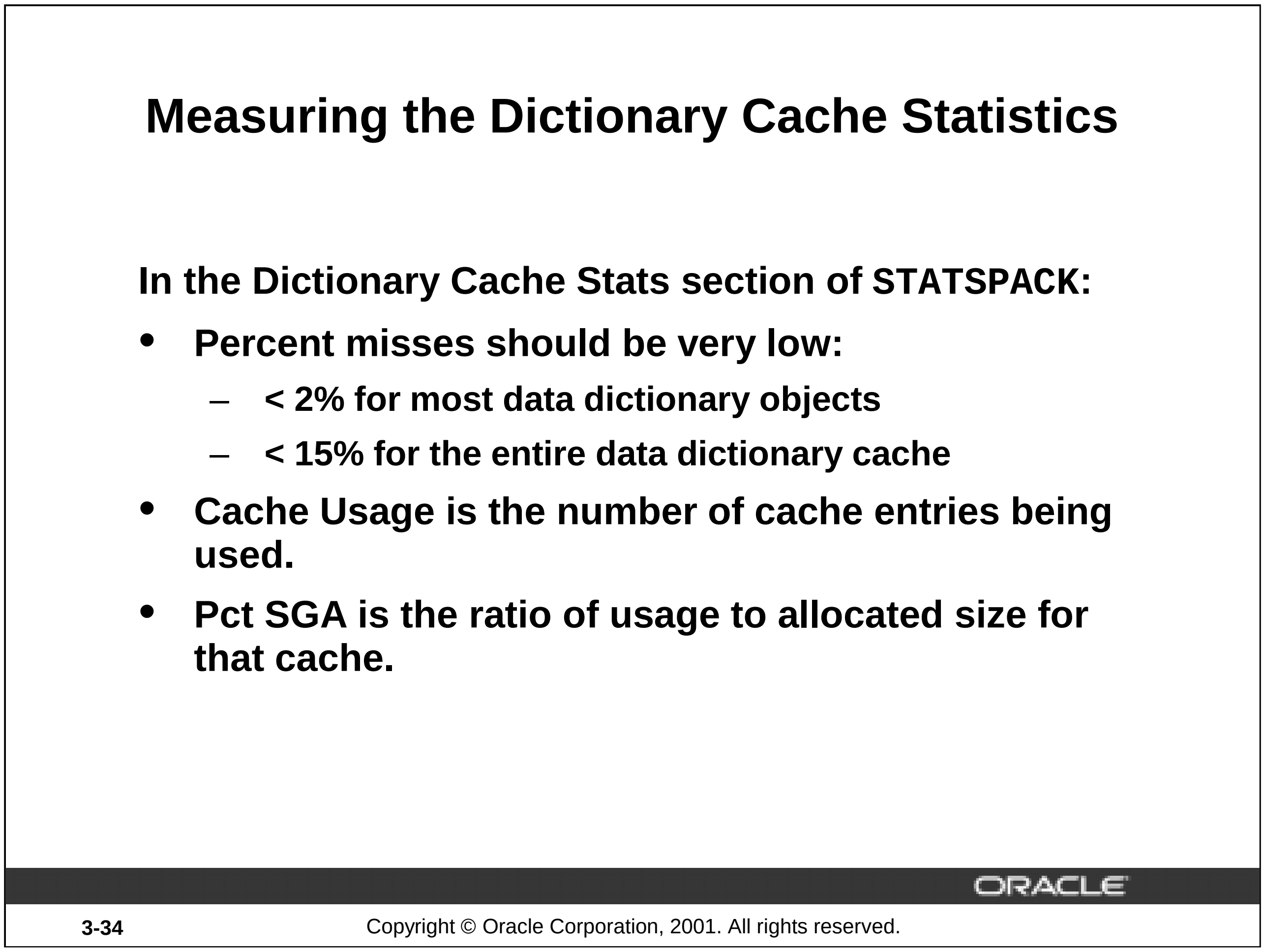
Monitoring the Dictionary Cache

Use the V$ROWCACHE view. The columns of most interest are shown in the following table:

Column	**Description**
PARAMETER	Categories of data dictionary items
GETS	Requests for information on that category
GETMISSES	Requests resulting in cache misses

Sizing

You can size the dictionary cache only indirectly with the SHARED_POOL_SIZE parameter. The algorithm for allocating shared pool space gives preference to the dictionary cache.



Measuring the Dictionary Cache Statistics

In the Dictionary Cache Stats section of STATSPACK:

- **Percent misses should be very low:**
  - **< 2% for most data dictionary objects**
  - **< 15% for the entire data dictionary cache**
- **Cache Usage is the number of cache entries being used.**
- **Pct SGA is the ratio of usage to allocated size for that cache.**

ORACLE

3-34

Measuring Dictionary Cache Statistics

The report by the STATSPACK utility contains a section on the statistics related to the dictionary cache. An example output of the section is here:

```
                         Get    Pct  Scan  Pct   Mod Final Pct
Cache                  Requests   Miss   Reqs  Miss  Reqs Usage  SGA
-------------------- -------- ------ ------ ----- ----- -----  ---
dc_free_extents           215   26.5     36   0.0   150    13  57
dc_histogram_defs          29   72.4      0           0   109  92
dc_object_ids             236    6.4      0           0   373  99
dc_objects                263    9.9      0           0   513  99
dc_profiles                 2    0.0      0           0     1  10
dc_rollback_segments       40    0.0      0           0     6  33
dc_segments               138   15.2      0          39   184  97
dc_tablespaces            123    0.0      0           0     6  86
dc_used_extents            57   63.2      0          57    38  67
dc_user_grants              6   16.7      0           0     5  16
dc_usernames              159    0.6      0           0     8  38
dc_users                   62    1.6      0           0     7  78
```

Measuring Dictionary Cache Statistics (continued)

```
dc_rollback_segments        40    0.0      0     0     6  33
dc_segments                138   15.2      0    39   184  97
dc_tablespaces             123    0.0      0     0     6  86
dc_used_extents             57   63.2      0    57    38  67
dc_user_grants               6   16.7      0     0     5  16
dc_usernames               159    0.6      0     0     8  38
dc_users                    62    1.6      0     0     7  78
```

Tuning the Data Dictionary Cache

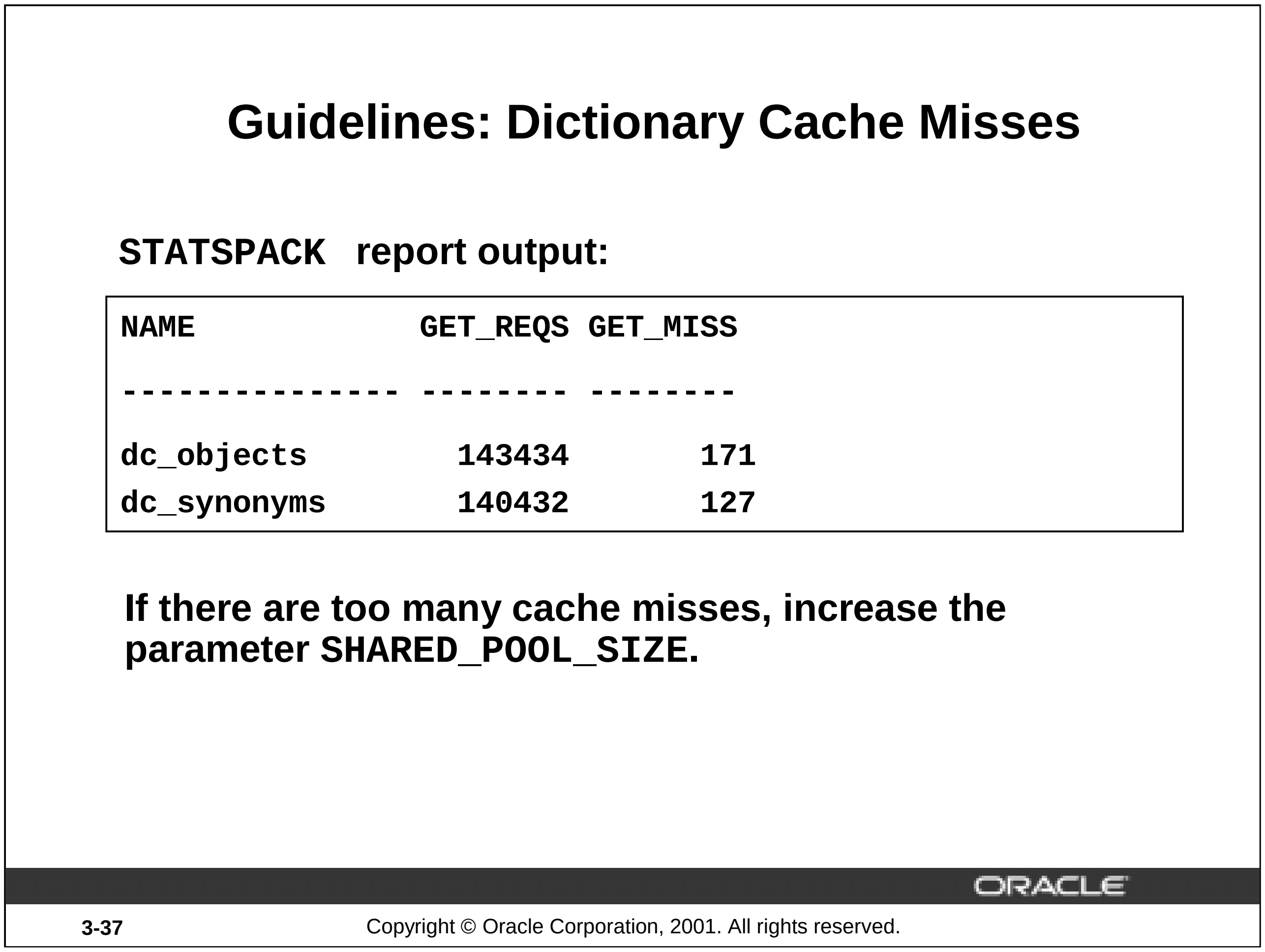
Keep the ratio of the sum of `GETMISSES` to the sum of `GETS` less than 15%:

```
SQL> select parameter, gets, getmisses
 2   from v$rowcache;
PARAMETER                  GETS      GETMISSES
-------------------------- --------- ---------
dc_objects                  143434       171
dc_synonyms                 140432       127
```

Goal for a Good Ratio

The ratio of the sum of all `GETMISSES` to the sum of all `GETS` should be less than 15% during normal running. If it is higher, consider increasing `SHARED_POOL_SIZE`.

You cannot hope to achieve a zero value for `GETMISSES`, because an object definition must be loaded into the cache the first time after startup that a server needs it.



Guidelines: Dictionary Cache Misses

`STATSPACK` report output:

```
NAME             GET_REQS GET_MISS

--------------- -------- --------

dc_objects        143434       171
dc_synonyms       140432       127
```

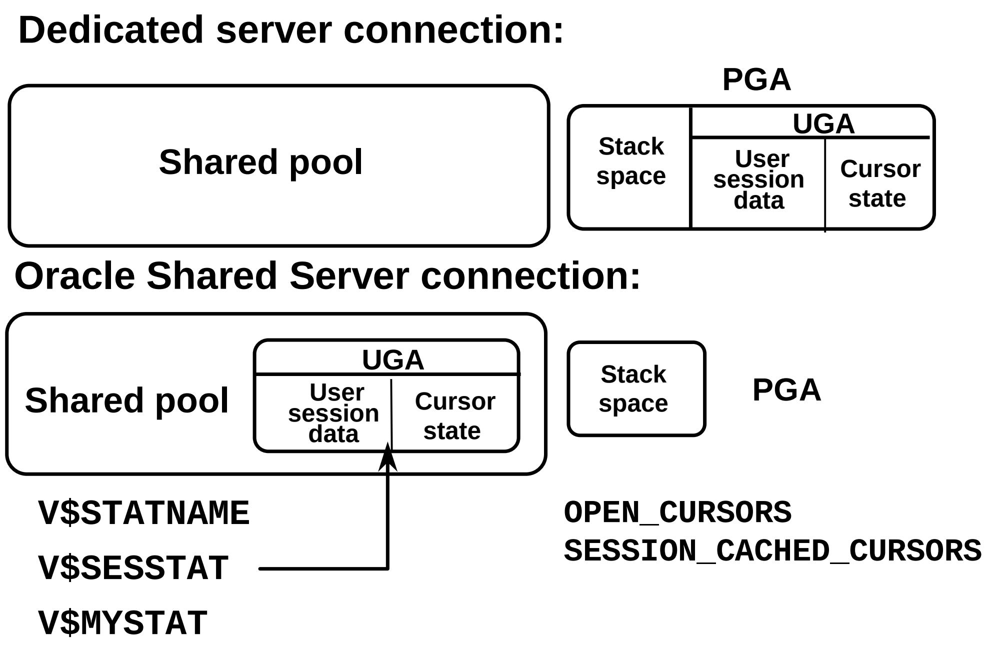
If there are too many cache misses, increase the parameter `SHARED_POOL_SIZE`.

3-37

Ratio from the `STATSPACK Report` Output

If the `STATSPACK` `report` output indicates a high `GET_MISS/GET_REQ` ratio for a number of items, the `SHARED_POOL_SIZE` should be increased.

UGA and Oracle Shared Server



The User Global Area

If you use the Oracle Shared Server, and the large pool is not configured, then user session and cursor state information is stored in the shared pool instead of in private user memory. Sort areas and private SQL areas are included in the session information. This is because shared servers work on a per-statement basis, so any server may need access to any user's information. This part of the shared pool is called the user global area (UGA).

The total memory requirement for the Oracle Shared Server is no larger than if you use dedicated servers. You may need to increase `SHARED_POOL_SIZE`, but your private user memory is lower.

Sizing the User Global Area

UGA space used by your connection:

```
SQL> select SUM(value) ||'bytes' "Total session memory"
  2  from V$MYSTAT, V$STATNAME
  3  where name = 'session uga memory'
  4  and v$mystat.statistic# = v$statname.statistic#;
```

UGA space used by all Oracle Shared Server users:

```
SQL> select SUM(value) ||'bytes' "Total session memory"
  2  from V$SESSTAT, V$STATNAME
  3  where name = 'session uga memory'
  4  and v$sesstat.statistic# = v$statname.statistic#;
```

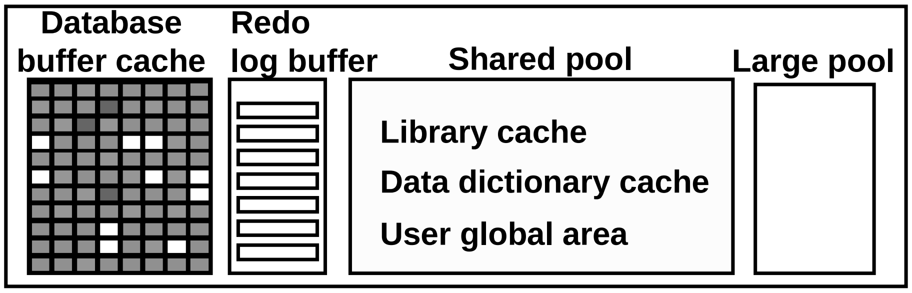
Maximum UGA space used by all users:

```
SQL> select SUM(value) ||'bytes' "Total max memory"
  2  from V$SESSTAT, V$STATNAME
  3  where name = 'session uga memory max'
  4  and v$sesstat.statistic# = v$statname.statistic#;
```

Required Space Measurement

For all Oracle Shared Server connections, you need to compute the amount of space required for all shared server users to put their session memory in the shared pool.

Large Pool



- Can be configured as a separate memory area in the SGA, used for memory with:
  - I/O server processes: `DBWR_IO_SLAVES`
  - Backup and restore operations
  - Session memory for the shared servers
  - Parallel query messaging
- Is useful in these situations to avoid performance overhead caused by shrinking the shared SQL cache
- Is sized by the parameter `LARGE_POOL_SIZE`

Existence of the Large Pool

The large pool must be explicitly configured. The memory of the large pool does not come out of the shared pool, but directly out of the SGA, thus adding to the amount of shared memory the Oracle server needs for an instance at startup.

Advantages of the Large Pool

The large pool is used to provide large allocations of session memory for:

- I/O server processes
- Backup and restore operations

The memory for backup and restore operations and for I/O server processes is allocated in buffers of a few hundred kilobytes. The large pool is better able to satisfy such requests than the shared pool.

Oracle Shared Server: By allocating session memory from the large pool for the Oracle Shared Server, the Oracle server can use the shared pool primarily for caching shared SQL and avoid the performance overhead caused by shrinking the shared SQL cache.

Summary

In this lesson, you should have learned how to:

- Size shared SQL and PL/SQL areas (library cache)
- Size data dictionary cache or row cache
- Size the large pool
- Size the user global area, if connections are Oracle Shared Server connections, unless the large pool is configured

3-41

Practice 3

The objective of this practice is to use diagnostic tools to monitor and tune the shared pool.

1. Connect using ‘sys/oracle as sysdba” and check the size of the shared pool.

   ```
   SQL> show parameter shared_pool
   ```

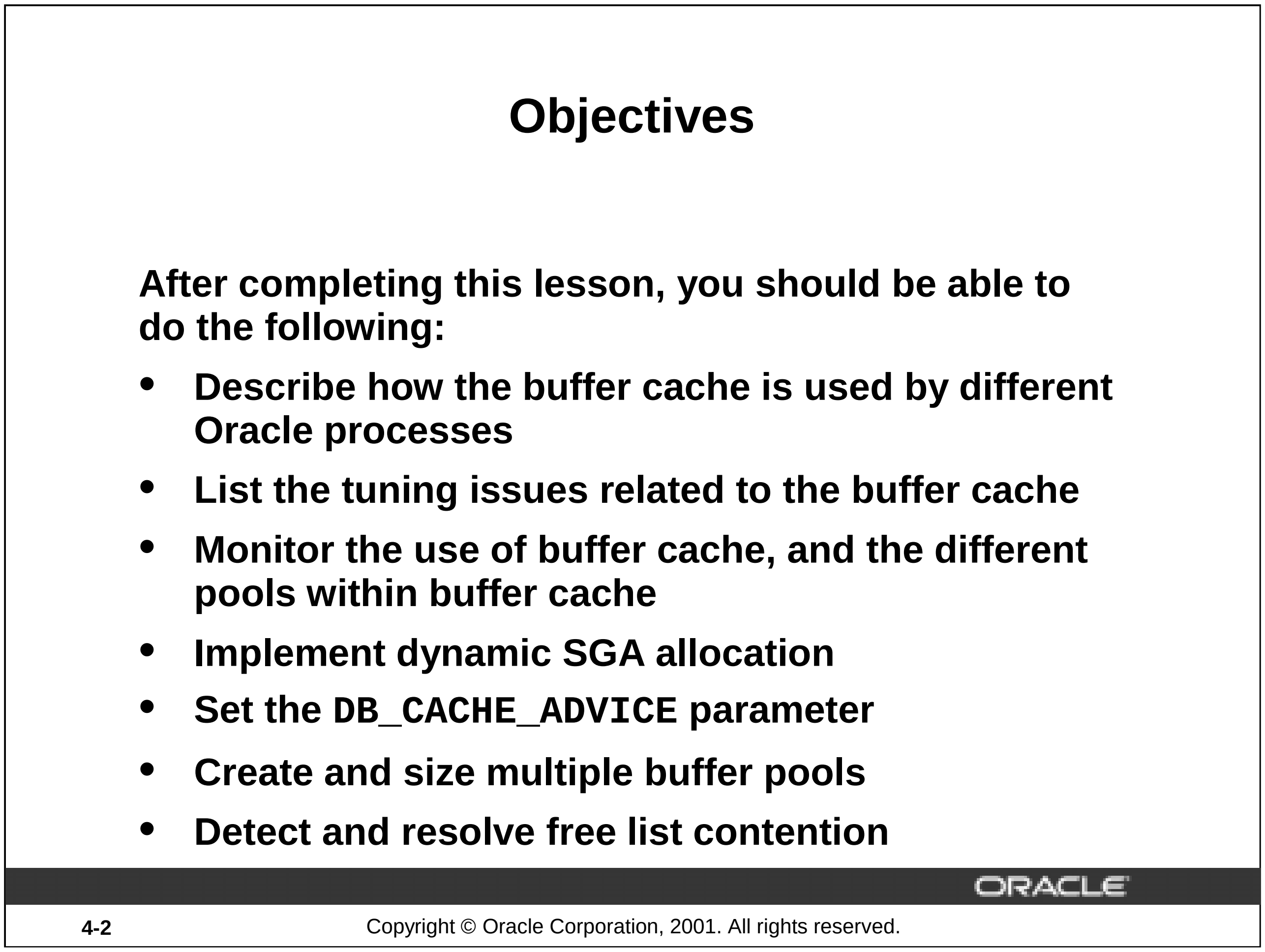
2. Connect as perfstat/perfstat user, execute the $HOME/STUDENT/LABS/snap.sql script to collect initial snapshot of statistics, and note the snapshot number by
3. To simulate user activity against the database open two operating system sessions. In session 1 connect as hr/hr and run the $HOME/STUDENT/LABS/lab03_03_1.sql script. In the second session connect as hr/hr and run the script $HOME/STUDENT/LABS/lab03_03_2.sql.
4. Connect as “system/manager” and measure the pin-to-reload ratio for the library cache by querying v$librarycache . Determine if it is a good ratio or not.

   Using the dynamic view:
5. Connect as “system/manager” and measure the get-hit ratio for the data dictionary cache by querying v$rowcache. Determine if it is a good ratio or not using the dynamic view.
6. Connect as perfstat/perfstat and run the script $HOME/STUDENT/LABS/snap.sql to collect a statistic snap shot and obtain the snapshot number. Record this number.
7. As user perfstat/perfstat obtain the statistics report between the two recorded snapshot_ids (from questions 2 and 6) by running the script $HOME/STUDENT/LABS/spreport.sql.
8. Analyze the generated report in the current directory (named sp_3_5.lst in the previous example). What would you consider to address if the library hit ratio (found under the heading “Instance Efficiency Percentages) is less than 98%?
9. Determine which packages, procedures, and triggers are pinned in the shared pool by querying v$db_object_cache.
10. Connect using “sys/oracle as sysdba” and pin one of the Oracle supplied packages that needs to be kept in memory, such as SYS.STANDARD using the procedure DBMS_SHARED_POOL.KEEP, that is created by running the script $ORACLE_HOME/rdbms/admin/dbmspool.sql.
11. Determine the amount of session memory used in the shared pool for your session by querying the v$mystat view. Limit the output by including the clause where name = ‘session uga memory’
12. Determine the amount of session memory used in the shared pool for all sessions, using V$SESSTAT and V$STATNAME views:

Quick Reference

Context	Reference
Initialization parameters	LARGE_POOL_SIZE PARALLEL_AUTOMATIC_TUNING SHARED_POOL_SIZE OPEN_CURSORS SESSION_CACHED_CURSORS CURSOR_SPACE_FOR_TIME SHARED_POOL_RESERVED_SPACE
Dynamic performance views	V$SGASTAT V$LIBRARYCACHE V$SQLAREA V$SQLTEXT V$DB_OBJECT_CACHE V$ROWCACHE V$SHARED_POOL_RESERVED V$STATNAME V$SESSTAT V$MYSTAT
Data dictionary views	None
Commands	None
Packaged procedures and functions	DBMS_SHARED_POOL.KEEP DBMS_SHARED_POOL.UNKEEP DBMS_SHARED_POOL.ABORTED_REQUEST_THRESHOLD
Scripts	dbmspool.sql prvtpool.plb
Oracle Diagnostics Pack	Performance Manager

4 Sizing the Buffer Cache

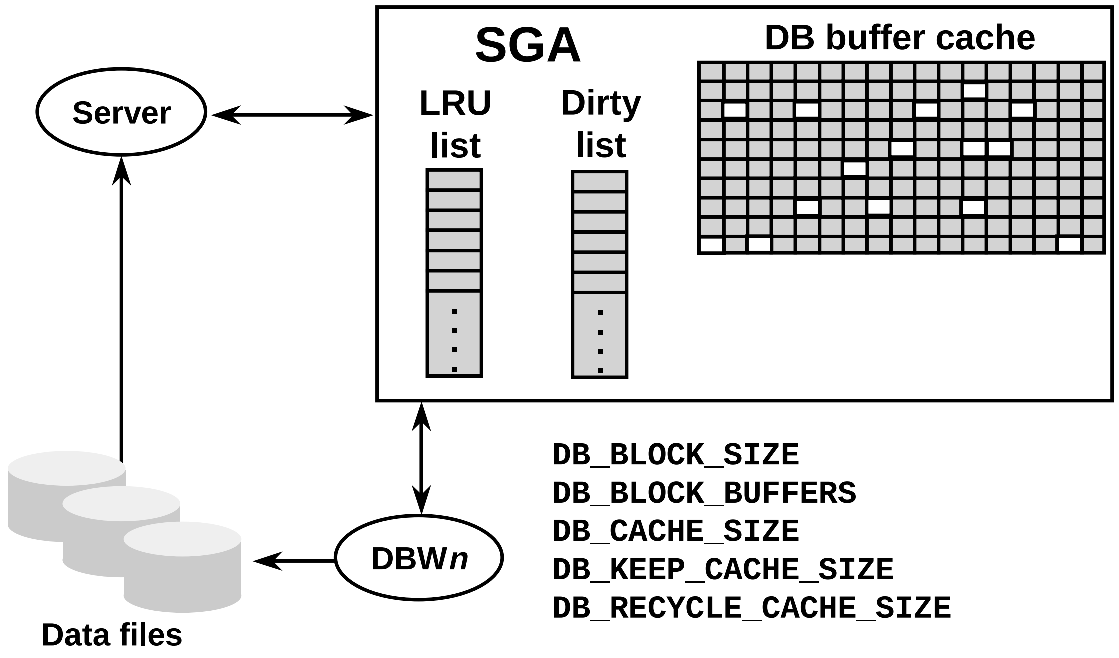


Objectives

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

- Describe how the buffer cache is used by different Oracle processes
- List the tuning issues related to the buffer cache
- Monitor the use of buffer cache, and the different pools within buffer cache
- Implement dynamic SGA allocation
- Set the `DB_CACHE_ADVICE` parameter
- Create and size multiple buffer pools
- Detect and resolve free list contention

Overview



DB_BLOCK_SIZE
DB_BLOCK_BUFFERS
DB_CACHE_SIZE
DB_KEEP_CACHE_SIZE
DB_RECYCLE_CACHE_SIZE

Buffer Cache Characteristics

The buffer cache holds copies of the data blocks from the data files. Because the buffer cache is a part of the SGA, these blocks can be shared by all users. The server processes read data from the data files into the buffer cache. To improve performance, the server process sometimes reads multiple blocks in a single read. The `DBWn` process writes data from the buffer cache into the data files. To improve performance, `DBWn` writes multiple blocks in a single write.

In Oracle9*i*, the buffer caches can be individually sized with the following parameters:

- `DB_CACHE_SIZE` specifies the size of default buffer pool in bytes.
- `DBA_KEEP_CACHE_SIZE` specifies the size of keep buffer pool in bytes.
- `DBA_RECYCLE_CACHE_SIZE` specifies the size of recycle buffer pool in bytes.

The buffer pools in Oracle9*i* can be resized dynamically.

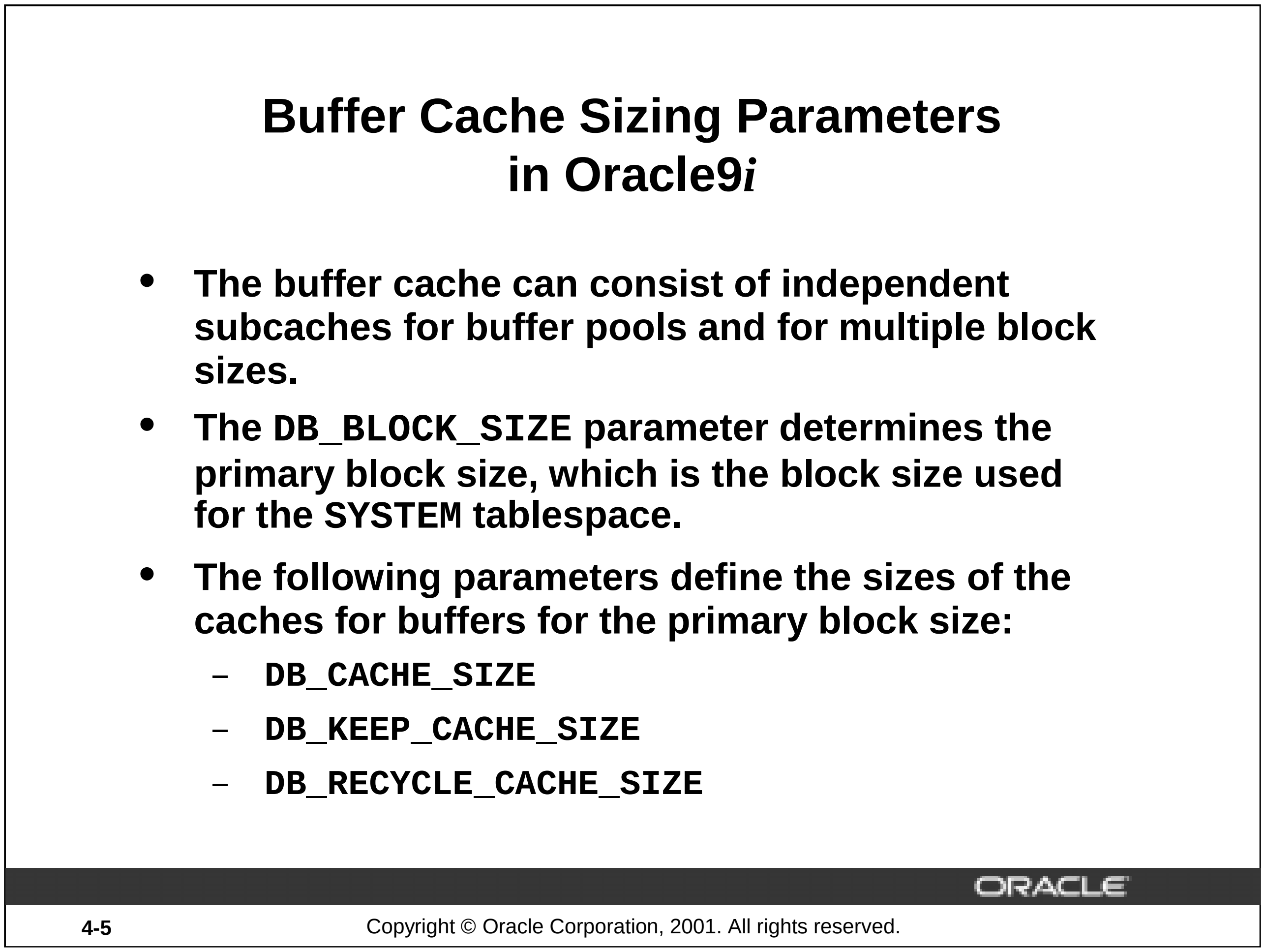
In Oracle8*i*, the buffer cache has the following characteristics.

- It is sized using the `DB_BLOCK_BUFFERS` parameter. This parameter specifies the number of blocks in the buffer cache. To find the size of the cache in bytes, multiply `DB_BLOCK_BUFFERS` by `DB_BLOCK_SIZE`.
- At any given time, the buffer cache may hold multiple copies of a single database block. Only one current copy of the block exists, but server processes may need to construct read-consistent copies, using rollback information, to satisfy queries.

Buffer Cache Characteristics (continued)

- The blocks in the buffer cache are managed using two lists:
  - The least recently used (LRU) list is used to keep the most recently accessed blocks in memory. The blocks on the list are organized from the most recently used (MRU) to the least recently used.
  - The dirty list points to blocks in the buffer cache that have been modified but not written to disk.
- Blocks in the buffer cache can be in one of three states:
  - Free buffers are blocks that have the same image on disk and in memory. These blocks are available for reuse.
  - Dirty blocks are blocks with a different image in memory than the image on disk. These blocks must be written to disk before they can be reused.
  - Pinned buffers are memory blocks that are currently being accessed.

Server processes use the blocks in the buffer cache, but the `DBWn` process makes blocks in the cache available by writing changed blocks back to the data files.



Buffer Cache Sizing Parameters in Oracle9*i*

- **The buffer cache can consist of independent subcaches for buffer pools and for multiple block sizes.**
- **The `DB_BLOCK_SIZE` parameter determines the primary block size, which is the block size used for the `SYSTEM` tablespace.**
- **The following parameters define the sizes of the caches for buffers for the primary block size:**
  - **`DB_CACHE_SIZE`**
  - **`DB_KEEP_CACHE_SIZE`**
  - **`DB_RECYCLE_CACHE_SIZE`**

4-5

Buffer Cache Size Parameters

- Oracle9*i* supports multiple block sizes. The block size, specified at the creation of the database for the `SYSTEM` tablespace, is referred to as the primary block size. Other tablespaces may be of different block size. You should concentrate on the tuning of the buffer cache relating to the primary block size.
- The size of the DEFAULT buffer pool for buffers with the primary block size is defined by the parameter `DB_CACHE_SIZE`. The KEEP and RECYCLE buffer pools may be configured for the primary block size using the `DB_KEEP_CACHE_SIZE` and `DB_RECYCLE_CACHE_SIZE` parameters, respectively.
- The values for the `DB_CACHE_SIZE, DB_KEEP_CACHE_SIZE`, and `DB_RECYCLE_CACHE_SIZE` parameters are specified in units of memory (KB or MB bytes), not in number of blocks.
- The quantum of cache defined by `DB_KEEP_CACHE_SIZE` and `DB_RECYCLE_CACHE_SIZE` does not come out of the default pool defined by `DB_CACHE_SIZE`. The values of these parameters are therefore independent of one another.

Dynamic SGA Feature in Oracle9*i*

- **The dynamic SGA feature implements an infrastructure to allow the server to change its SGA configuration without shutting down the instance.**
- **With a dynamic SGA, the Oracle server can modify its physical address space use to respond to the operating system’s use of physical memory.**
- **A dynamic SGA provides an SGA that will grow and shrink in response to a DBA command.**

ORACLE

Dynamic SGA

The SGA has always been a static allocation of memory, which was shared across all threads of execution. Beginning with Oracle9*i*, the dynamic SGA infrastructure allows for the sizing of the buffer cache, shared pool, and large pool without having to shut down the instance, modify the initialization parameter file, and restart the instance. In addition, the dynamic SGA infrastructure allows limits to be set at run time on how much physical memory is used for the SGA. You should configure instances to start with less than the maximum amount of memory the operating system makes available, and allow the instances to grow as needed.

Unit of Allocation in the Dynamic SGA

- **In the dynamic SGA model, a new unit of allocation called a granule has been created.**
- **SGA memory is tracked in granules by SGA components.**
- **Use `V$BUFFER_POOL` to monitor granule allocation and de-allocation in the buffer cache.**

ORACLE

Unit of Allocation in the Dynamic SGA

The columns in `V$BUFFER_POOL` are:

- `BLOCK_SIZE`: Block size for buffers in this component
- `PRIMARY`: Primary block size (Y or N)
- `BUFFER_POOL`: Buffer pool this cache represents. Possible values are DEFAULT, KEEP, RECYCLE, or NULL (for nonprimary block sizes)
- `TOTAL_SIZE`: Present size of the component, in MB
- `BUFFER`: Present number of buffers in this component
- `TARGET_SIZE`: If a resize is in progress, records new target size; otherwise it is null
- `LO_SETNUM`: Low working set number for this component
- `HI_SETNUM`: High working set number for this component

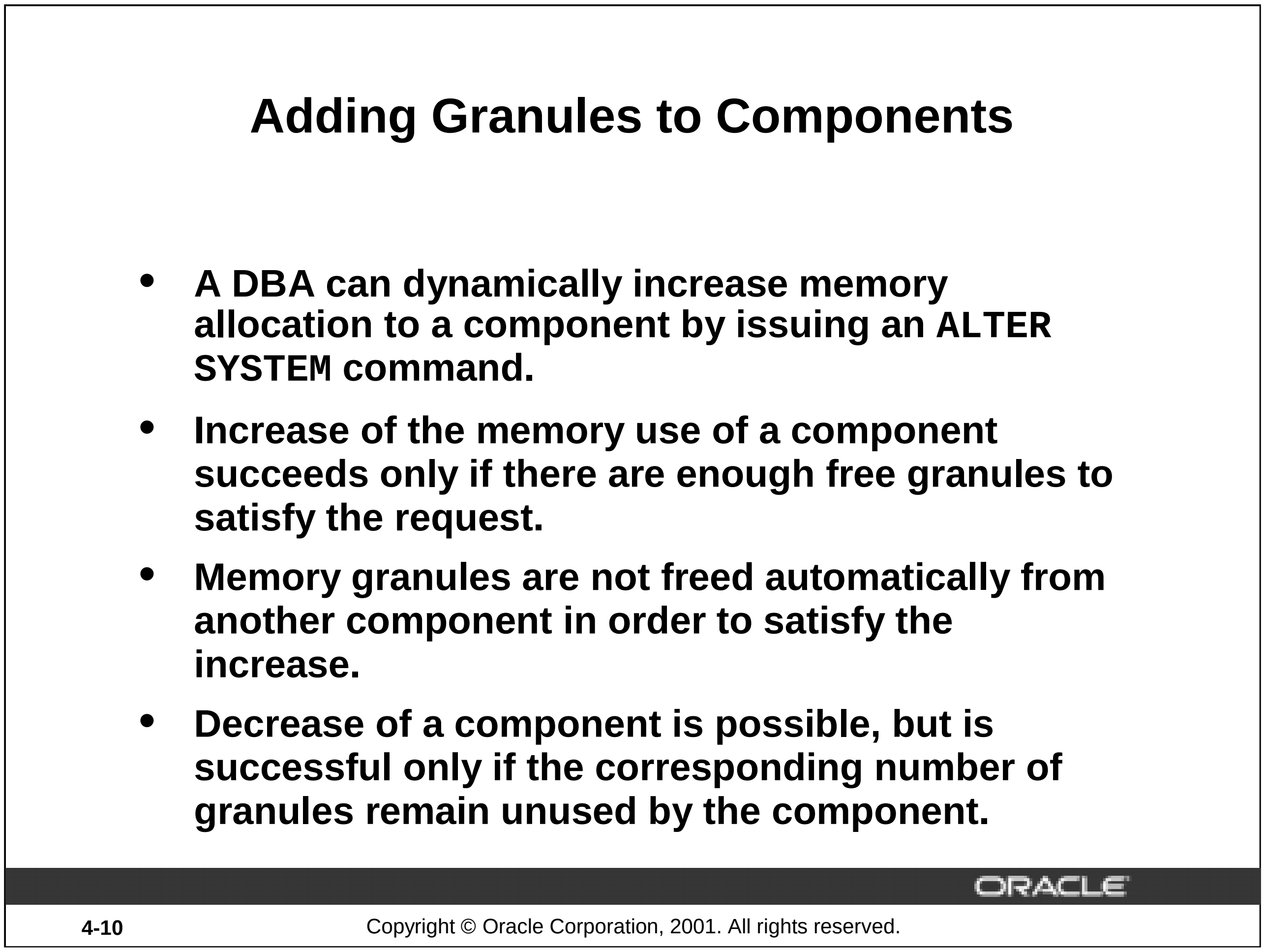
Granule

- **In Oracle9*i*, SGA components are allocated and deallocated in units of contiguous memory called granules.**
- **A granule is a unit of contiguous virtual memory allocation.**
- **The size of a granule depends on the estimated total SGA:**
  - **4 MB if estimated SGA size is less than 128 MB**
  - **16 MB otherwise**

Allocating Granules at Startup

- At instance startup, the Oracle server allocates granule entries, one for each granule to support `SGA_MAX_SIZE` bytes of address space.
- As startup continues, each component acquires as many granules as it requires.
- The minimum SGA configuration is three granules:
  - One granule for fixed SGA (includes redo buffers)
  - One granule for the buffer cache
  - One granule for the shared pool

4-9



Adding Granules to Components

- A DBA can dynamically increase memory allocation to a component by issuing an `ALTER SYSTEM` command.
- Increase of the memory use of a component succeeds only if there are enough free granules to satisfy the request.
- Memory granules are not freed automatically from another component in order to satisfy the increase.
- Decrease of a component is possible, but is successful only if the corresponding number of granules remain unused by the component.

4-10 Copyright © Oracle Corporation, 2001. All rights reserved.

```
SQL> show parameter db_cache_size;
NAME                                TYPE        VALUE
------------------------------- ----------- ------------------
db_cache_size                        big integer 4194304

SQL> alter system set db_cache_size=8M;
System altered.
SQL> show parameter db_cache_size;
NAME                                   TYPE     VALUE
------------------------------- ----------- ------------------
db_cache_size                       big integer 8388608
```

Dynamic Buffer Cache Size Parameters

- **Parameters that specify the size of buffer cache components are dynamic, and can be changed while the instance is running by means of the `ALTER SYSTEM` command:**

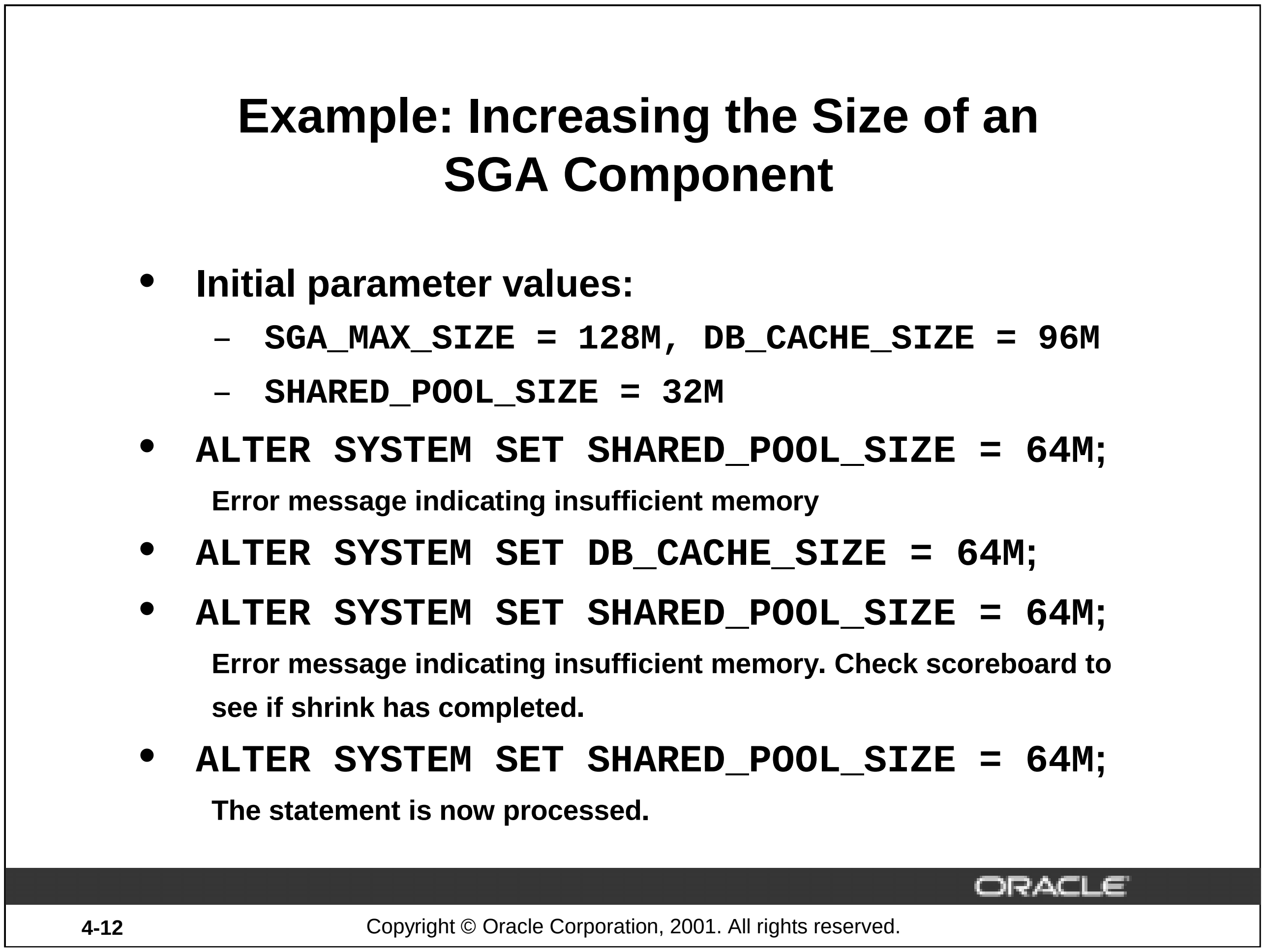
```
ALTER SYSTEM SET DB_CACHE_SIZE = 1100M;
```

- **Each parameter is sized independently.**
- **New cache sizes are set to the next granule boundary.**
- **The allocation size has the following limits:**
  - **It must be an integer multiple of the granule size.**
  - **The total SGA size cannot exceed `MAX_SGA_SIZE`.**
  - **`DB_CACHE_SIZE` can never be set to zero.**

4-11

Dynamic Buffer Cache Size Parameters

The buffer cache sizing parameters are now dynamic. They can be changed while the instance is running using the `ALTER SYSTEM` command.



Example: Increasing the Size of an SGA Component

- **Initial parameter values:**
  - **`SGA_MAX_SIZE = 128M, DB_CACHE_SIZE = 96M`**
  - **`SHARED_POOL_SIZE = 32M`**
- **`ALTER SYSTEM SET SHARED_POOL_SIZE = 64M;`**

  Error message indicating insufficient memory
- **`ALTER SYSTEM SET DB_CACHE_SIZE = 64M;`**
- **`ALTER SYSTEM SET SHARED_POOL_SIZE = 64M;`**

  Error message indicating insufficient memory. Check scoreboard to see if shrink has completed.
- **`ALTER SYSTEM SET SHARED_POOL_SIZE = 64M;`**

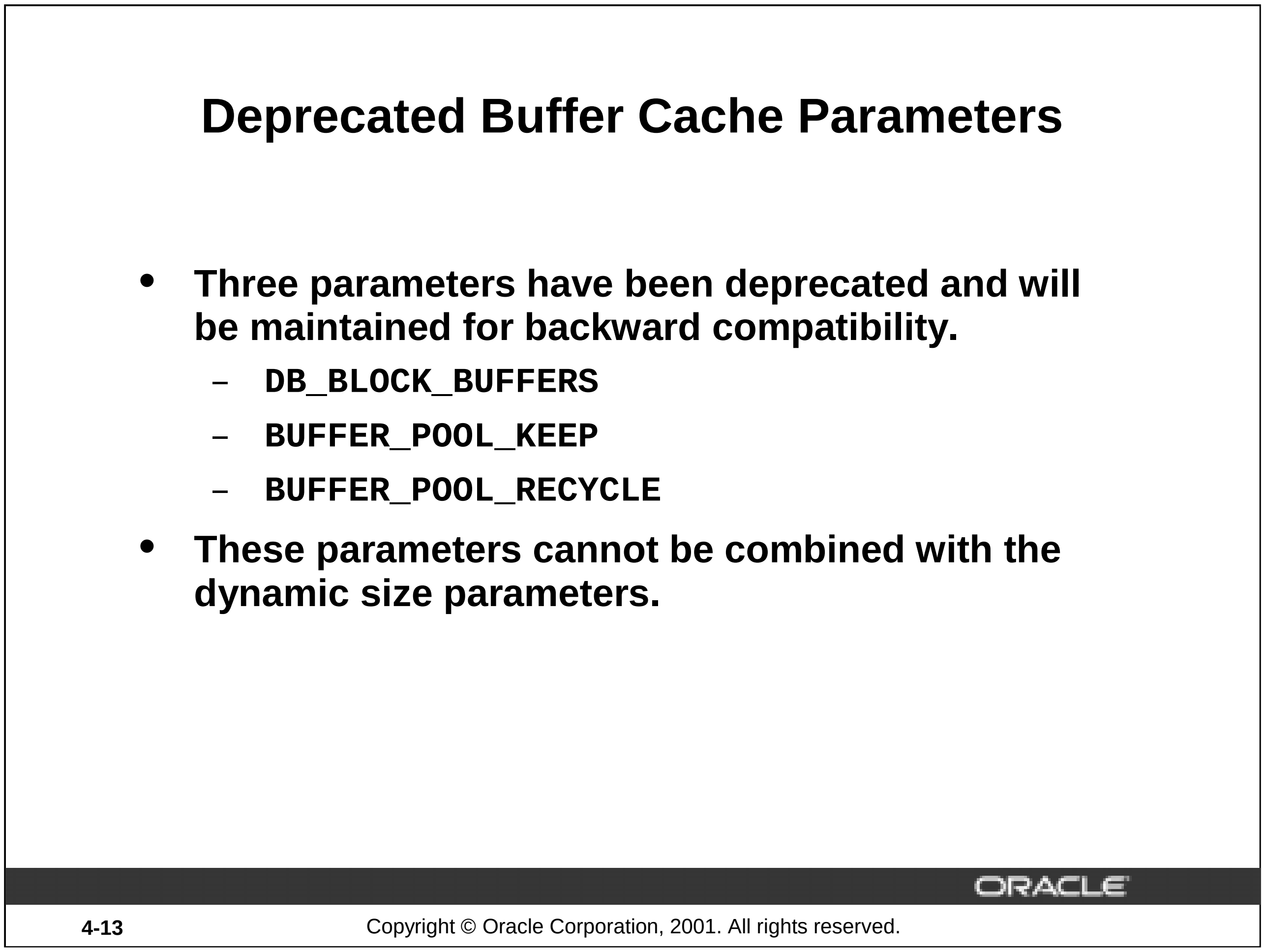
  The statement is now processed.

4-12 Copyright © Oracle Corporation, 2001. All rights reserved.

Increasing the Size of an SGA Component

A database administrator can increase the size of any SGA component by issuing an `ALTER SYSTEM` command. The new size is rounded up to the nearest multiple of the granule size.

Increasing the memory use of a component with an `ALTER SYSTEM` command succeeds if there are enough free granules (`SGA_ MAX_ SIZE` – current `SGA_SIZE`) to satisfy the request. The server does not start freeing another component's granules for adding. Instead, the database administrator must ensure the instance has enough free granules to satisfy the increase of a component's granule use. If the current SGA memory is less than `SGA_MAX_SIZE`, then the server is free to allocate more granules until the SGA size reaches `SGA_MAX_SIZE`.



Deprecated Buffer Cache Parameters

- **Three parameters have been deprecated and will be maintained for backward compatibility.**
  - **`DB_BLOCK_BUFFERS`**
  - **`BUFFER_POOL_KEEP`**
  - **`BUFFER_POOL_RECYCLE`**
- **These parameters cannot be combined with the dynamic size parameters.**

4-13

Deprecated Buffer Cache Parameters

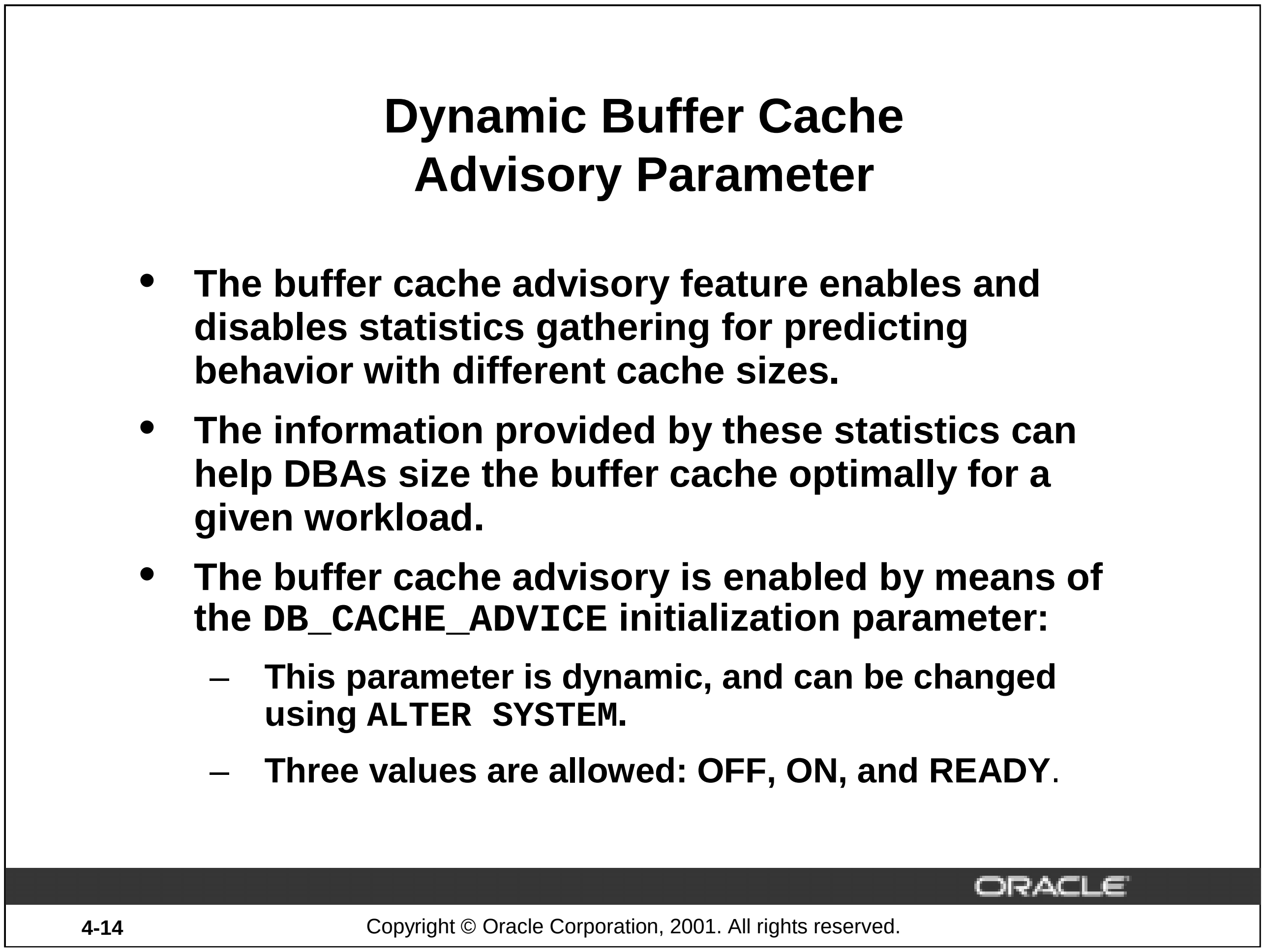
The present buffer cache size parameters `DB_BLOCK_BUFFERS`, `BUFFER_POOL_KEEP`, and `BUFFER_POOL_RECYCLE` are maintained for backward compatibility (so users can continue to use pre-Oracle9*i* parameter files), but are deprecated and will be made obsolete in the future.

The syntax for the `BUFFER_POOL_SIZE` parameters allowed the user to optionally specify the number of LRU latches for the buffer pool in addition to the number of buffers in the buffer pool. These latches were allocated out of the total number of latches specified in `DB_BLOCK_LRU_LATCHES`. Because `DB_BLOCK_LRU_LATCHES` is now obsolete, the specification of the number of LRU latches for the buffer pool, if provided, is ignored.

These parameters will continue to be static parameters. Furthermore, these parameters cannot be combined with the dynamic size parameters.

If these parameters are combined with new parameters at the start of the instance, you would get an error as follows:

```
ORA-00381: cannot use both new and old parameters for buffer
cache size specification
```



Dynamic Buffer Cache Advisory Parameter

- **The buffer cache advisory feature enables and disables statistics gathering for predicting behavior with different cache sizes.**
- **The information provided by these statistics can help DBAs size the buffer cache optimally for a given workload.**
- **The buffer cache advisory is enabled by means of the `DB_CACHE_ADVICE` initialization parameter:**
  - **This parameter is dynamic, and can be changed using `ALTER SYSTEM`.**
  - **Three values are allowed: OFF, ON, and READY**.

4-14

Dynamic Buffer Cache Advisory Parameter Values

- OFF: Advisory is turned off and the memory for the advisory is not allocated
- READY: Advisory is turned off, but the memory for the advisory remains allocated. Allocating the memory before the advisory is actually turned on avoids the risk of an ORA-4031 error (inability to allocate from the shared pool). If the parameter is switched to this state from OFF, an ORA-4031 error may be generated.
- ON: Advisory is turned on and both CPU and memory overhead is incurred. Attempting to set the parameter to this state when it is in the OFF state may lead to an ORA-4031 error when the parameter is switched to ON. If the parameter is in READY state it can be set to ON without error because the memory is already allocated.

View to Support Buffer Cache Advisory

- Buffer cache advisory information is collected in the `V$DB_CACHE_ADVICE` view.
- The view contains different rows that predict the estimated number of physical reads for different cache sizes.
- The rows also compute a physical read factor, which is the ratio of the number of estimated reads to the number of reads actually performed during the measurement interval by the real buffer cache.

View to Support Buffer Cache Advisory

`V$DB_CACHE_ADVICE` columns:

- `ID`: Buffer pool ID (ranges from 1–8)
- `NAME`: Buffer pool name
- `BLOCK_SIZE`: Block size in bytes for buffers in this pool. Possible values are the standard block size, and non-standard block sizes in powers of two: 2048, 4096, 8192, 16384, or 32768.
- `ADVICE_STATUS`: Status of the advisory
- `SIZE_FOR_ESTIMATE`: Cache size for prediction (in megabytes)
- `BUFFERS_FOR_ESTIMATE`: Cache size for prediction (in terms of buffers)
- `ESTD_PHYSICAL_READ_FACTOR`: Physical read factor for this cache size; ratio of number of estimated physical reads to the number of reads in the real cache. If there are no physical reads into the real cache, the value of this column is null.
- `ESTD_PHYSICAL_READS`: Estimated number of physical reads for this cache size

Using V$DB_CACHE_ADVICE

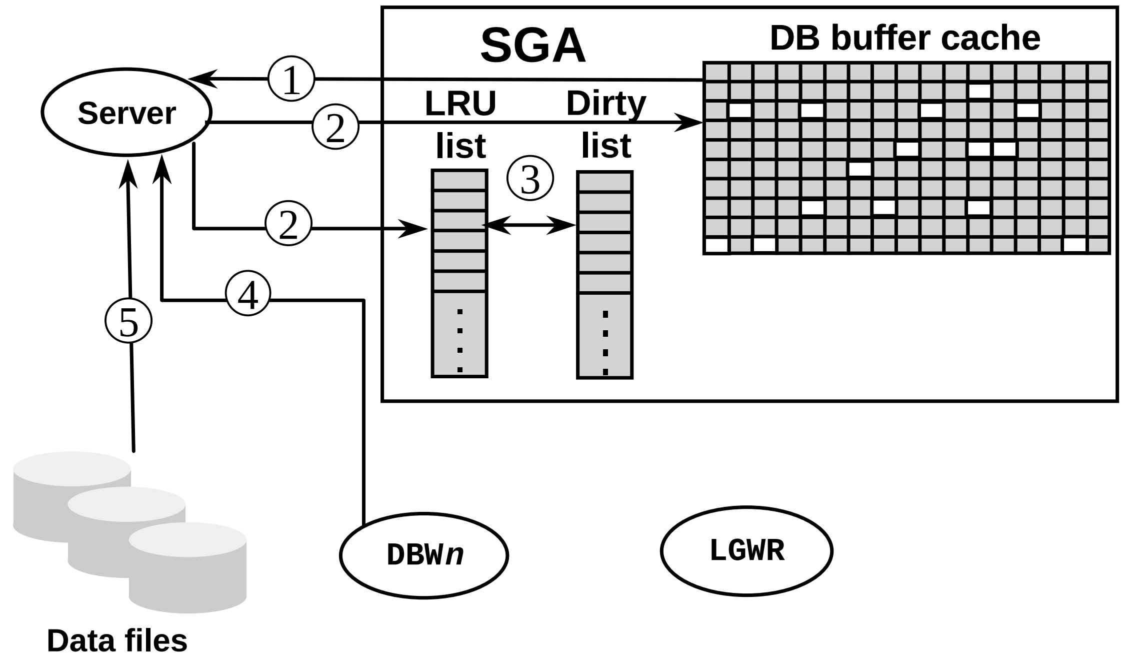
```
SELECT size_for_estimate,
  buffers_for_estimate,
  estd_physical_read_factor,
  estd_physical_reads
FROM V$DB_CACHE_ADVICE
  WHERE name = 'DEFAULT‘
  AND block_size = (
      SELECT value FROM V$PARAMETER
      WHERE name = 'db_block_size')
  AND advice_status = 'ON';
```

Using V$DB_CACHE_ADVICE View

The following output shows that if the cache was 212MB, rather than the current size of 304MB, the estimated number of physical reads would be 17 million (17,850,847). Increasing the cache size beyond its current size would not provide a significant benefit.

```
                                       Estd Phys    Estd Phys
 Cache Size (MB)      Buffers Read Factor        Reads
---------------- ------------ ----------- ------------
(10%)         30        3,802       18.70  192,317,943
              60        7,604       12.83  131,949,536
              91       11,406        7.38   75,865,861
             121       15,208        4.97   51,111,658
             152       19,010        3.64   37,460,786
             182       22,812        2.50   25,668,196
             212       26,614        1.74   17,850,847
             243       30,416        1.33   13,720,149
             273       34,218        1.13   11,583,180
(Current)    304       38,020        1.00   10,282,475
             334       41,822         .93    9,515,878
             364       45,624         .87    8,909,026
             395       49,426         .83    8,495,039
             424       53,228         .79    8,116,496
(150%)       456       57,030         .76    7,824,764
…
```

Managing the Database Buffer Cache

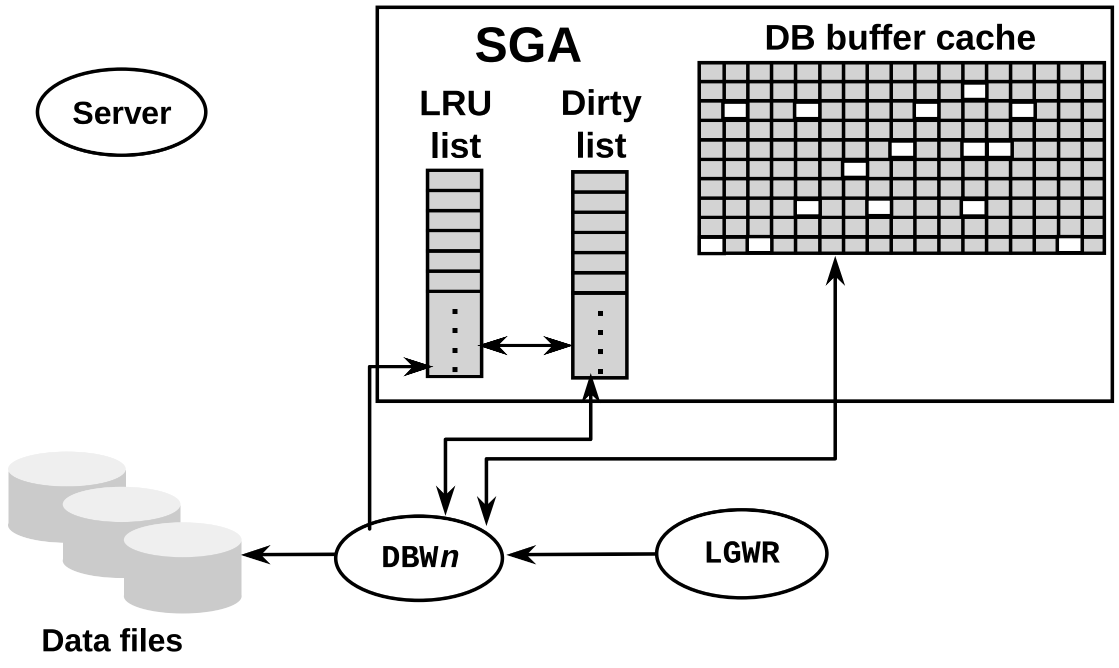


The Server Process and the Database Buffer Cache

When a server needs a block, it follows these steps to read the block:

1. First, the server checks whether the required block is available in the buffer cache using a hash function. If the block is found, it is moved to another point in LRU list away from the LRU end. This is a logical read, because no actual I/O took place. If the block is not found in the buffer cache, the server process has to read the block from the data file.
2. Before reading from the data file, the server process searches the LRU list for a free block.
3. While searching the LRU list, the server process moves dirty blocks to the dirty list.
4. If the dirty list exceeds its size threshold, the server signals `DBWn` to flush dirty blocks from the data buffer cache. If the server cannot find a free block within a search threshold, it signals `DBWn` to flush.
5. After a free block is found, the server reads the block from the data file into the free block in the database buffer cache. Oracle server process moves the block away from the LRU end of the LRU list.
6. If the block is not consistent, the server rebuilds an earlier version of the block from the current block and rollback segments.

Managing the Database Buffer Cache



The DBW*n* Process and the Database Buffer Cache

DBW*n* manages the buffer cache by writing dirty blocks to the data files to ensure that there are free blocks for servers. DBW*n* responds to different events in the instance.

7. Dirty List Exceeds its Size Threshold: A server process finds that the dirty list has exceeded its size threshold, so it signals DBW*n* to flush. DBW*n* writes out the blocks on the dirty list.
8. Search Threshold Exceeded: A server process that cannot find a free block on the LRU list within the search threshold signals DBW*n* to flush dirty blocks. DBW*n* writes out dirty blocks directly from the LRU list.
9. Three Second Time-Out: Every three seconds, DBW*n* checks the dirty list for blocks to write. DBW*n* moves dirty blocks from the LRU list to the dirty list, so that it has enough blocks for a full write buffer. Then DBW*n* writes blocks on the dirty list from the buffer cache to the data files. If there is no update activity for extended periods of time, DBW*n* eventually writes out all of the dirty blocks during the three-second time-outs.
10. LGWR Signals a Checkpoint: When LGWR signals that a checkpoint has occurred, DBW*n* copies dirty blocks from the LRU to the dirty list and writes out the blocks on the dirty list.

The DBW*n* Process and the Database Buffer Cache (continued)

11. `Alter Tablespace Offline Temporary or Alter Tablespace Begin Backup:` When a tablespace is altered `offline temporary` or its online backup is started, `DBW`*`n`* copies the dirty blocks for that tablespace from the LRU to the dirty list (step 8) and writes out the blocks on the dirty list (steps 6 and 7).
12. Drop Object: When an object is dropped, `DBW`*`n`* first flushes the objects dirty blocks to disk (steps 8 and 7).
13. Clean Shutdown (Normal, Immediate, or Transactional)

Tuning Goals and Techniques

- **Tuning goals:**
  - **Servers find data in memory**
  - **90% hit ratio for OLTP**
- **Diagnostic measures**
  - **Cache hit ratio**
  - **`V$DB_CACHE_ADVICE`**
- **Tuning techniques:**
  - **Increase buffer cache size**
  - **Use multiple buffer pools**
  - **Cache tables**
  - **Bypass the buffer cache for sorting and parallel reads**

Tuning Goals

Because physical I/O takes significant time and increases CPU demand, Oracle server performance is improved when the servers find most of the blocks that they need in memory. The statistic that measures the performance of the database buffer cache is the cache hit ratio. This statistic is the ratio of the number of blocks found in memory to the number of blocks accessed. When the database buffer cache is too small, the system is slower because it is performing too many I/Os.

Diagnostic Measures

To effectively monitor the usage of the buffer cache, you can use the following mechanisms:

- Measure the cache hit ratio: Use the `V$SYSSTAT` view, the `utlbstat.sql` and `utlestat.sql` scripts, or the Stats Pack utility (available in Oracle8*i* and later versions)
- Use the `V$DB_CACHE_ADVICE` view (available in Oracle9*i*)

Tuning Techniques

The DBA monitors the buffer cache by calculating the cache hit ratio from statistics collected by the Oracle server.

Tuning Techniques (continued)

To improve the cache hit ratio, the DBA can:

- Increase the size of buffer cache
- Use multiple buffer pools to separate blocks by access characteristics
- Configure the tables to be cached in memory
- Configure to bypass the buffer cache for sorting and parallel reads if possible.

First, the DBA determines the change in the hit ratio as buffers are added or removed. As a general rule, increase buffer cache size if:

- The cache hit ratio is less than 90%
- There is adequate memory for other processes without inducing additional page faults

Increasing the size of the data buffer cache does not always improve performance. The characteristics of the application may prevent further improvement of the cache hit ratio. For example, in large data warehouse or decision support systems, which routinely use many scans of large tables, most of the data is read from disk. For such systems, tuning the buffer cache is less important and tuning I/O is vital.

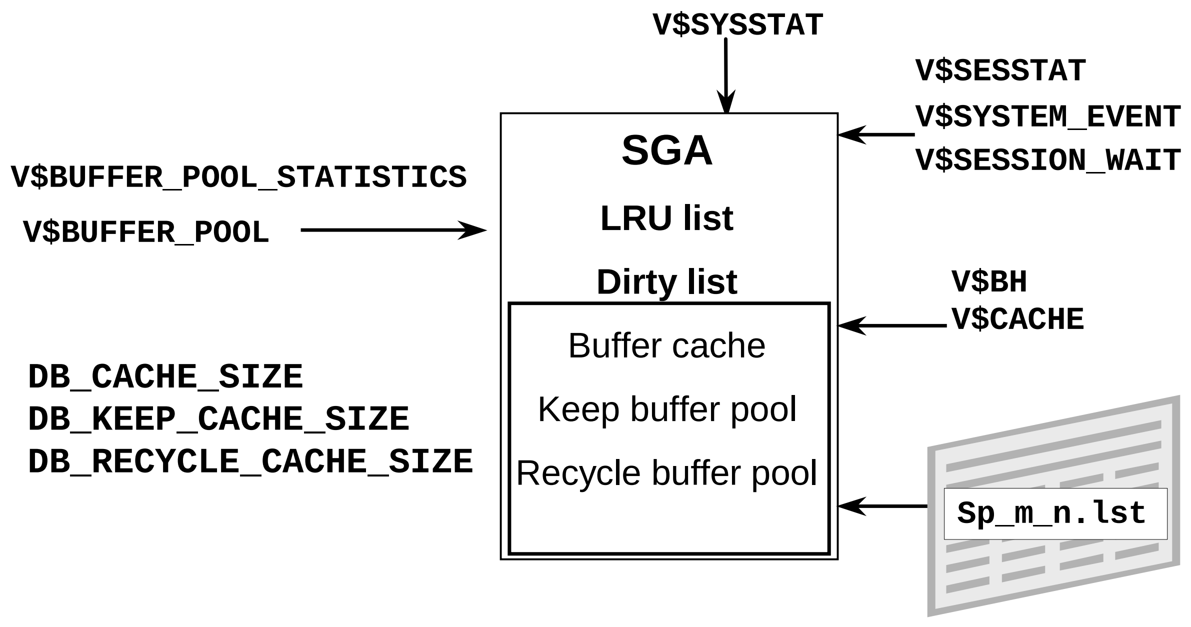
If data access characteristics are causing the low cache hit ratio, the DBA may be able to improve the ratio by defining multiple pools or caching tables.

Technical Note

You need to consider the impact of operating system caching. For example, the Oracle server may show a high rate of physical I/O that does not appear at the operating system level. This could mean that Oracle blocks, aged out of the buffer cache, are kept in the operating system cache and can be accessed very quickly. However, as a general rule it is best to bypass the operating system cache, because:

- More memory may be required to maintain duplicate blocks in memory (one block in the operating system cache and one in the database buffer cache)
- There is the CPU overhead of copying blocks from the operating system cache to the database buffer cache

Diagnostic Tools



Description of the Views

- The V$SYSSTAT and V$SESSTAT views contain the statistics used to calculate the cache hit ratio:

```
                SQL> SELECT  name, value  FROM v$sysstat
                                    2>            WHERE
name in (‘session logical reads‘,
    3>                  ‘physical reads`,
    4>                  ‘physical reads direct’
    5>                  ‘physical reads direct (lob)’);
```

- V$BUFFER_POOL: Describes multiple buffer pools, and V$BUFFER_POOL STATISTICS shows information on individual pool. You can also use monitor buffer pools with the query:

```
SQL> SELECT name, physical_reads, db_block_gets,
2>          consistent_gets
3>   FROM   v$buffer_pool_statistics;
```

- V$BH: Describes blocks held in the buffer cache

Oracle8*i* parameters that define the buffer cache are:
DB_BLOCK_SIZE, DB_BLOCK_BUFFERS, DB_BLOCK_LRU_LATCHES, BUFFER_POOL_KEEP, and BUFFER_POOL_RECYCLE.

Measuring the Cache Hit Ratio

From V$SYSSTAT:

```
SQL> SELECT 1 - (phy.value – lob.value – dir.value)
   / ses.value  "CACHE HIT RATIO"
  2  FROM   v$sysstat ses, v$sysstat lob,
  3         v$sysstat dir, v$sysstat phy
  3  WHERE  ses.name = 'session logical reads'
  4  AND    dir.name = ‘physical reads direct'
  5  AND    lob.name = 'physical reads direct (lob)'
  6  AND    phy.name = 'physical reads';
```

From the STATSPACK report:

```
Statistic                   Total      Per        Per
                            Trans    Logon     Second
--------------------- ---------------   ---------
physical reads         15,238      13.0   15,238.0
physical reads direct     863       0.7      863.0
Physical reads direct(lob)  0         0          0
session logical reads 119,376     101.8  119,376.0
```

Measuring the Cache Hit Ratio

The server collects statistics on data access and stores them in the dynamic performance table `V$SYSSTAT`. You measure the cache hit ratio using the following system statistics:

- `physical reads`: Number of blocks read from disk
- `physical reads direct`: Number of direct reads, does not require the cache
- `physical reads direct (lob)`: Number of direct reads of large binary objects
- `session logical reads:` Number of logical read requests

Calculate the hit ratio for the buffer cache with this formula:

Hit Ratio = 1 – (physical reads – physical reads direct - physical reads direct (lob)) / session logical reads

Session logical reads gives the total number of read requests for data. This value includes requests satisfied by access to buffers in memory and requests that cause a physical I/O. You can multiply the ratio by 100 to convert it to a percentage.

Because these statistics are collected since the instance startup time, query them during normal working loads but not immediately after startup. Because the buffer cache is empty when the instance starts, there are more physical reads after startup.

Guidelines for Using the Cache Hit Ratio

Hit ratio is affected by data access methods:

- **Full table scans**
- **Data or application design**
- **Large table with random access**
- **Uneven distribution of cache hits**

Guidelines to Increase the Cache Size

Increase the cache size ratio under the following conditions:

- **The cache hit ratio is less than 0.9 on OLTP system.**
- **There is no undue page faulting.**
- **If the previous increase was effective.**

Guidelines

Increasing the Cache Hit Ratio by Reducing Buffer Cache Misses

If your hit ratio is low, you may want to increase the number of buffers in the cache to improve performance.

To make the buffer cache larger:

- Allocate more memory to the buffer cache by decreasing the size of any other component of SGA with unused memory.
- If there is room for SGA to grow (that is, if `SGA_MAX_SIZE` is not reached), use `ALTER SYSTEM` to increase the value of `DB_CACHE_SIZE,` (`DB_KEEP_CACHE_SIZE`, or `DB_RECYCLE_CACHE_SIZE`).
- Increase the value of the `DB_BLOCK_BUFFERS` initialization parameter for oracle8*i* databases. This parameter is static.

The DBA then collects new statistics that estimate the performance gain resulting from increasing the size of the buffer cache. With these statistics, you can estimate how many buffers to add to your cache if necessary.

Removing Unnecessary Buffers when Cache Hit Ratio Is High

If your hit ratio is high, your cache is probably large enough to hold your most frequently accessed data. In this case, you can reduce the cache size and still maintain good performance.

Guidelines (continued)

Removing Unnecessary Buffers When Cache Hit Ratio Is High

To make the buffer cache smaller, reduce the value of the `DB_BLOCK_BUFFERS` initialization parameter. The minimum value for this parameter is 4. You can use any leftover memory for other Oracle memory structures.

The DBA then collects new statistics to calculate buffer cache performance based on a smaller cache size. Examining these statistics can help you determine how small you can afford to make your buffer cache without adversely affecting performance.

Evaluating the Cache Hit Ratio

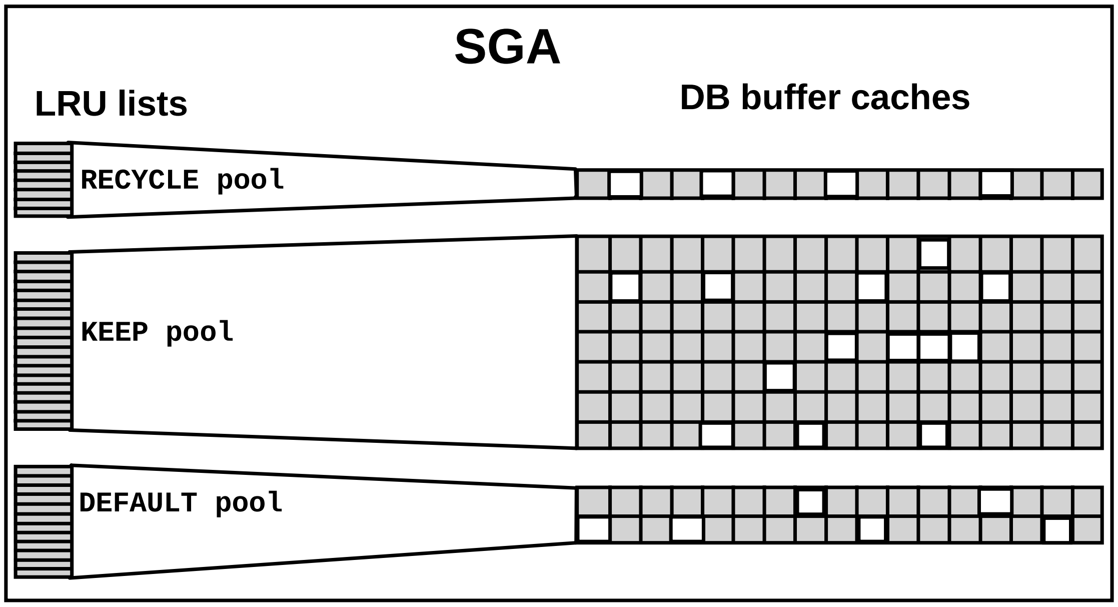
- Do not continue increasing `DB_BLOCK_BUFFERS` if the last increase made no significant difference in the cache hit ratio. This may be because of the way that you are accessing your data, or there may be other operations that do not even use the buffer pool. For example, the Oracle server bypasses the buffer cache for sorting and parallel reads.
- Also, when looking at the cache hit ratio, bear in mind that blocks encountered during a full table scan are not put to the head of the LRU list; therefore, repeated scanning does not cause the blocks to be cached.
- The solution lies at the design or implementation level, in that repeated scanning of the same large table is rarely the most efficient solution to the problem. It may be better to perform all of the processing in a single pass or add appropriate indexes to the table.
- In large databases running an OLTP application, most rows are accessed either one or zero times in any given unit of time. On this basis there is little point in keeping the row (or the block that contains it) in memory for very long after its use.
- Finally, the relationship between cache hit ratio and number of buffers is far from a smooth distribution. When tuning the buffer pool, avoid the use of additional buffers that contribute little or nothing to the cache hit ratio.

Increasing `DB_BLOCK_BUFFERS`

As a general rule, increase `DB_BLOCK_BUFFERS` under the following conditions:

- The cache hit ratio is less than 90%.
- There is adequate memory for other processes, as measured by the amount of page faults.
- The previous increase of `DB_BLOCK_BUFFERS` was effective.

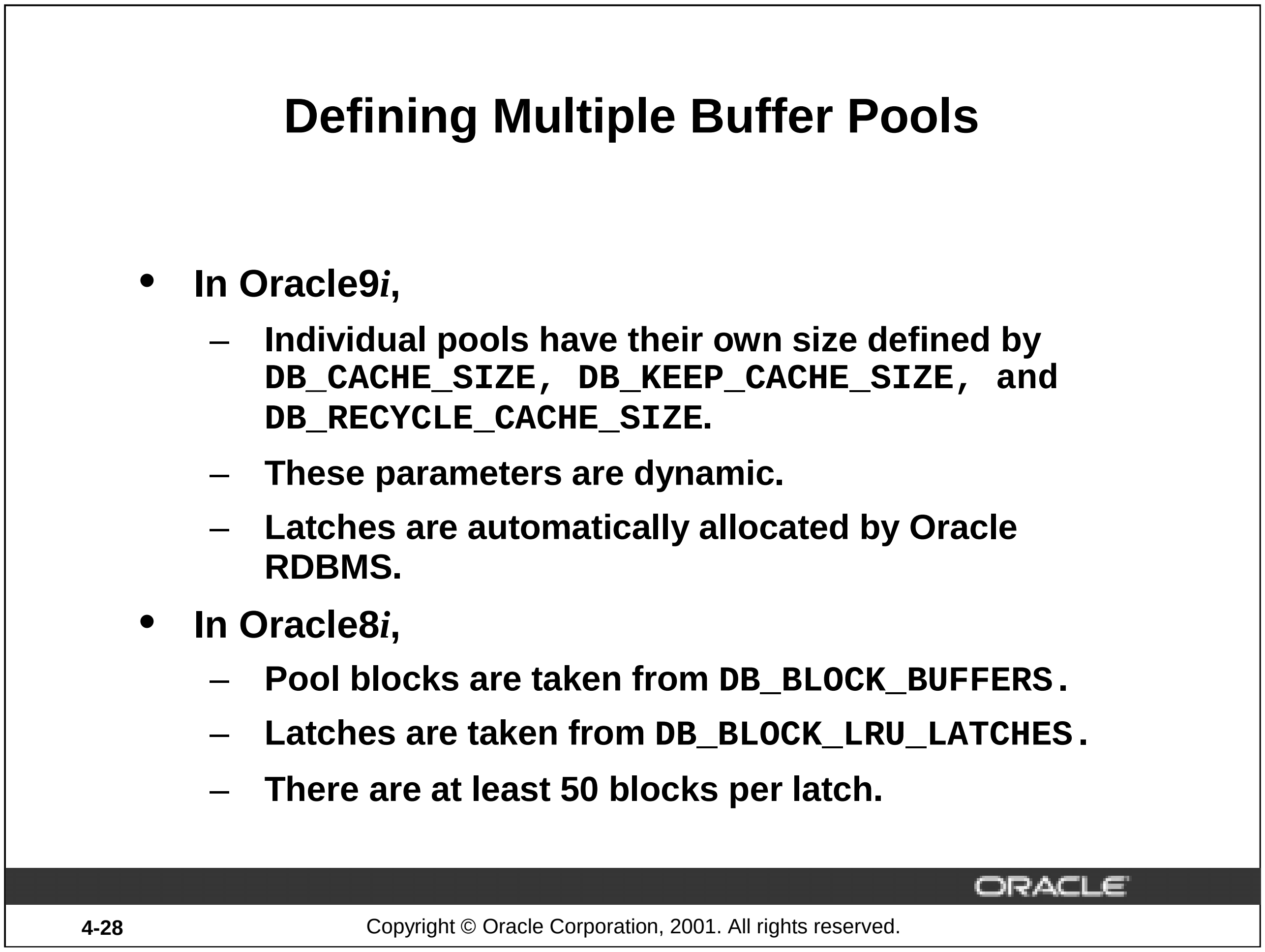
Using Multiple Buffer Pools



Multiple Buffer Pools

The DBA may be able to improve the performance of the database buffer cache by creating multiple buffer pools. Objects are assigned to a buffer pool depending on how the objects are accessed. There are three buffer pools:

- KEEP: This pool is used to retain objects in memory that are likely to be reused. Keeping these objects in memory reduces I/O operations.
- RECYCLE: This pool is used to eliminate blocks from memory that have little chance of being reused. Flushing these blocks from memory enables you to allocate the space that would be used by their cache buffers to other objects.
- DEFAULT: The pool always exists. It is equivalent to the single buffer cache.



Defining Multiple Buffer Pools

- **In Oracle9*i*,**
  - **Individual pools have their own size defined by `DB_CACHE_SIZE`, `DB_KEEP_CACHE_SIZE`, and `DB_RECYCLE_CACHE_SIZE`.**
  - **These parameters are dynamic.**
  - **Latches are automatically allocated by Oracle RDBMS.**
- **In Oracle8*i*,**
  - **Pool blocks are taken from `DB_BLOCK_BUFFERS`.**
  - **Latches are taken from `DB_BLOCK_LRU_LATCHES`.**
  - **There are at least 50 blocks per latch.**

Defining Multiple Buffer Pools

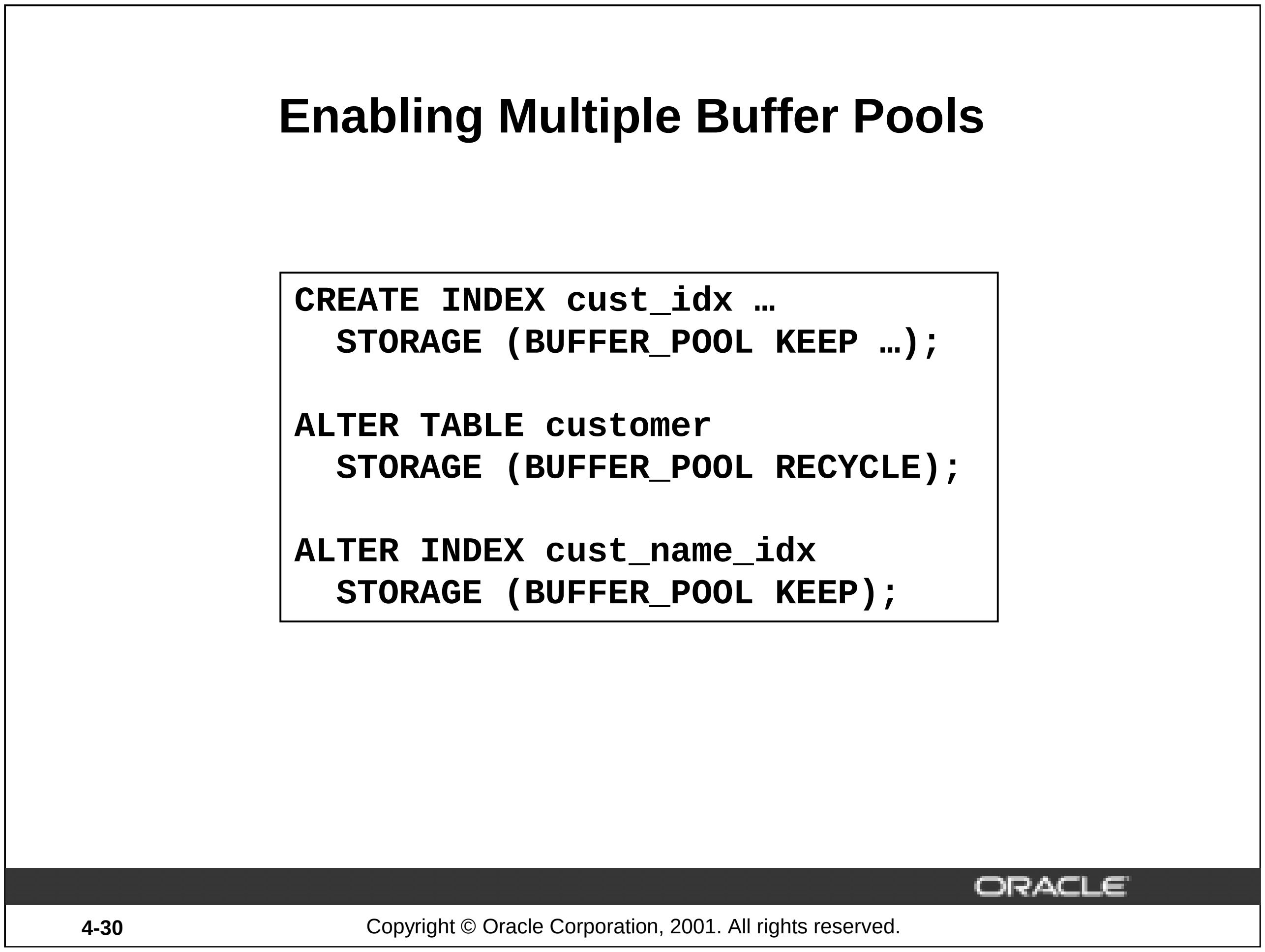
You can define three buffer pools: Default, Recycle, and Keep. The fact that such blocks are assigned to KEEP pool does not mean that Oracle process will retain them in the cache for a longer period of time. You should segregate such tables by using the storage clause on the underlying tables. By such segregation, you are managing to avoid contention for blocks of different (say RECYCLE or DEFAULT) pools.

Initialization Parameters:

- `DB_CACHE_SIZE:` Defines the number of buffers for the default pool in Oracle9*i*. In Oracle9*i*, this pool is individually configured and other pools do not become a part of this pool. In Oracle8*i*, `DB_BLOCK_BUFFERS` parameter defines the size of the buffer cache for the instance. In Oracle8*i*, each individual buffer pool is created from this total amount; the remainder is allocated to the default buffer pool.
- `DB_KEEP_CACHE_SIZE:` (In Oracle8*i*, `BUFFER_POOL_KEEP`) Defines the buffer pool blocks of data to be retained in cache. In Oracle9*i*, the memory blocks for the KEEP pool are independent of the `DB_CACHE_SIZE`, whereas in Oracle8*i*, memory for KEEP pool is allocated from that defined in `DB_BLOCK_BUFFERS`.
- `DB_RECYCLE_CACHE_SIZE: (`In Oracle8*i*, `BUFFER_POOL_RECYCLE)`Defines the buffer pool for blocks that may not be retained in memory for long. As already stated, the RECYCLE pool gets its allocation from the `DB_BLOCK_BUFFERS` in Oracle8*i*.

Defining Multiple Buffer Pools (continued)

- `DB_BLOCK_LRU_LATCHES`: In Oracle8*i*, this parameter is used to allocate the number of LRU latches for the entire database instance (each defined buffer pool takes a latch from this total). In Oracle9*i*, this parameter is not available.
- The minimum number of buffers that must be allocated to each buffer pool is 50 times the number of LRU latches. For example, if a buffer pool has three LRU latches, it must have at least 150 buffers.There is no requirement that any buffer pool be defined for another buffer pool to be used.



Enabling Multiple Buffer Pools

```
CREATE INDEX cust_idx …
  STORAGE (BUFFER_POOL KEEP …);

ALTER TABLE customer
  STORAGE (BUFFER_POOL RECYCLE);

ALTER INDEX cust_name_idx
  STORAGE (BUFFER_POOL KEEP);
```

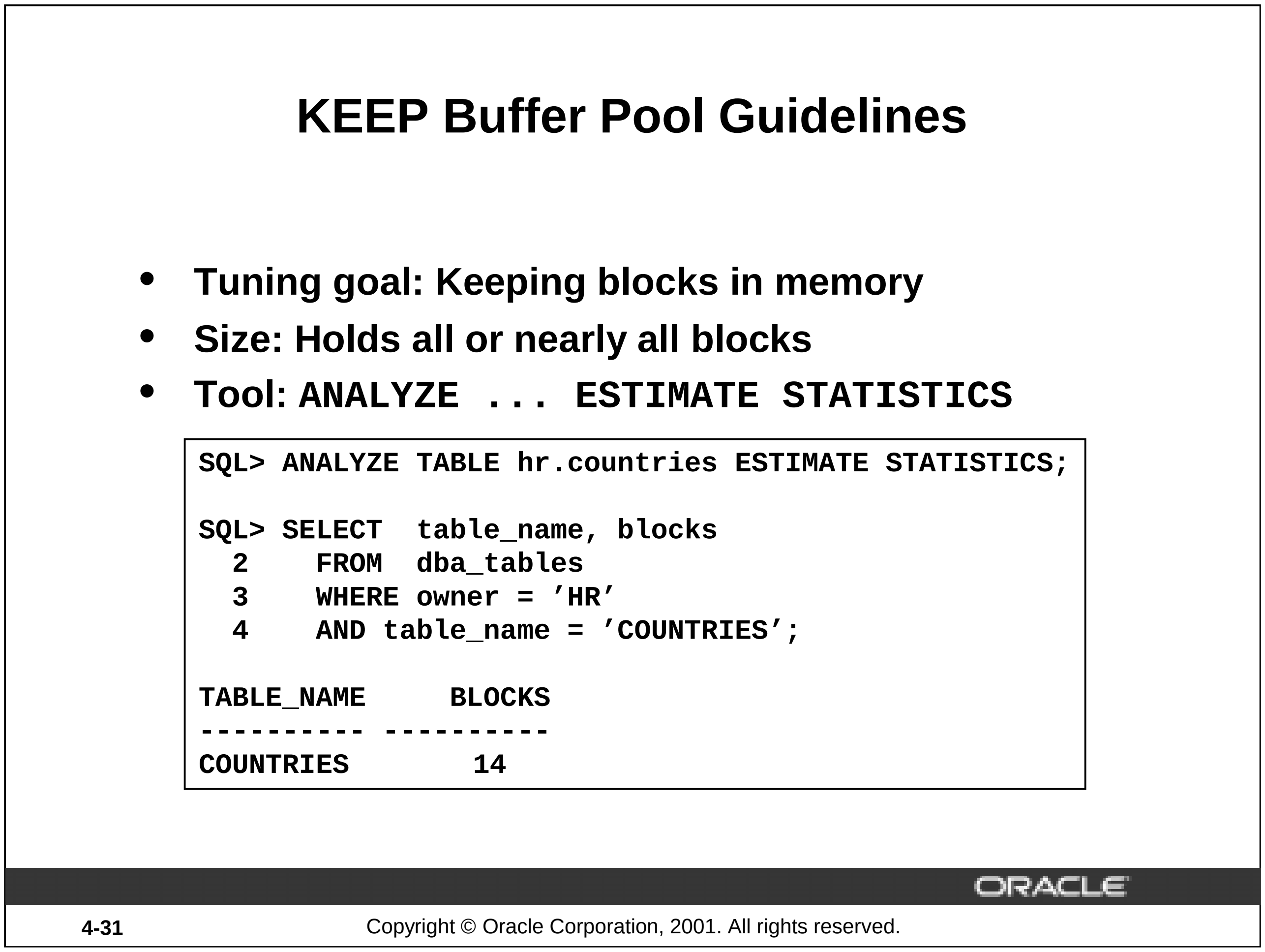
The BUFFER_POOL Clause

The `BUFFER_POOL` clause is used to define the default buffer pool for an object. It is part of the `STORAGE` clause and is valid for `CREATE` and `ALTER` table, cluster, and index statements. The blocks from an object without an explicitly set buffer pool go into the DEFAULT buffer pool.

The syntax is `BUFFER_POOL { KEEP | RECYCLE | DEFAULT }.`

When the default buffer pool of an object is changed using the `ALTER` statement, all buffers currently containing blocks of the altered segment remain in the buffer pool they were in before the `ALTER` statement. Newly loaded blocks, and any blocks that have been aged out and reloaded, go into the new buffer pool.

Because buffer pools are assigned to a segment, objects with multiple segments can have blocks in multiple buffer pools. For example, an index-organized table can have different pools defined on both the index and the overflow segment.



KEEP Buffer Pool Guidelines

- **Tuning goal: Keeping blocks in memory**
- **Size: Holds all or nearly all blocks**
- **Tool: `ANALYZE ... ESTIMATE STATISTICS`**

```
SQL> ANALYZE TABLE hr.countries ESTIMATE STATISTICS;

SQL> SELECT  table_name, blocks
  2    FROM  dba_tables
  3    WHERE owner = 'HR'
  4    AND table_name = 'COUNTRIES';

TABLE_NAME     BLOCKS
---------- ----------
COUNTRIES       14
```

Tuning Goal

The goal of the keep buffer pool is to retain objects in memory, thus avoiding I/O operations. The size of the keep buffer pool is computed by adding together the sizes of all objects dedicated to this pool.

Sizing

Use `ANALYZE ... ESTIMATE STATISTICS` to obtain the size of each object. Indexes may also be put in the keep pool, where necessary. Use ANALYZE INDEX VALIDATE STRUCTURE to find the number of leaf and branch blocks used by the index.

The high-water mark is always exact, even if you estimate statistics. Add up the `BLOCKS` column from `DBA_TABLES`, `DBA_TAB_PARTITIONS`, `DBA_INDEXES`, and `DBA_CLUSTERS` to get the total blocks required.

Depending on the data access characteristics and the amount of available memory, you may not want to keep all of the blocks from all of these objects in the buffer pool. Often you can significantly decrease the size of your KEEP buffer pool and still maintain a high hit ratio. Those blocks can be allocated to other buffer pools.

The DBA must monitor objects in the KEEP pool that grow in size. An object may no longer fit in the KEEP buffer pool, in which case you will begin to lose blocks out of the cache.

RECYCLE Buffer Pool Guidelines

- **Tuning goal: Eliminating blocks from memory when transactions are completed**
- **Size: Holds only active blocks**
- **Tool: V$CACHE**

```
...

SQL> SELECT  owner#, name, count(*) blocks
  2    FROM  v$cache
  3    GROUP BY owner#, name;

OWNER# NAME           BLOCKS
------ ---------- ----------
     5 CUSTOMER          147
...
```

Tuning Goal

The goal of the recycle buffer pool is to eliminate blocks from memory as soon as they are no longer needed. Be careful, however, not to discard blocks from memory too quickly. If the buffer pool is too small, it is possible for a block to age out of the cache before the transaction or SQL statement has completed execution. For example, an application may select a value from a table, use the value to process some data, and then update the selected row. If the block is removed from the cache after the `SELECT` statement, it must be read from disk again to perform the update. The block needs to be retained for the duration of the transaction.

Sizing

You can size the recycle pool by using the physical reads statistic from a tracing tool or by totaling the buffer cache blocks used by the object.

The view `V$CACHE` must be created using the `catparr.sql` in the SYS schema.

Tuning Goal (continued)

Using `V$CACHE` to Find Blocks in the Buffer Pool

The DBA can also monitor the number of buffer pool blocks by object using `V$CACHE`. `V$CACHE` is created by the `catparr.sql` script.

`V$CACHE`:

- Is intended for use with Oracle Parallel Server (OPS)
- Creates a number of other views that are useful only for OPS
- Maps extents in the data files to database objects
- Needs to be rerun after new objects have been created

To determine the number of blocks required for objects in the RECYCLE pool:

- Tune the buffer cache with the RECYCLE pool disabled.
- Run `catparr.sql` to set up and populate `V$CACHE`.
- During peak running times, use the following query to calculate how many blocks are used by each object:

```
SQL> SELECT  owner#, name, count(*) blocks
 2   FROM  v$cache
 3   GROUP BY owner#, name;
```

- Sum the blocks for all objects that will be used in the RECYCLE buffer pool and divide by four to get RECYCLE pool size. You divide by four because it is assumed that one-fourth of the blocks targeted for the RECYCLE pool are active; the other three-fourths are waiting to be aged out of the cache.

RECYCLE Buffer Pool Guidelines

Tool: V$SESS_IO

```
SQL> SELECT s.username,
  2         io.block_gets,
  3         io.consistent_gets,
  4         io.physical_reads
  5  FROM   v$sess_io     io,
  6         v$session     s
  6  WHERE  io.sid  = s.sid ;

BLOCK_GETS CONSISTENT_GETS PHYSICAL_READS
---------- --------------- --------------
      2187           23271           1344
```

Tracing Physical Reads

By executing statements with a SQL statement tuning tool such as Oracle Trace Manager, SQL*Plus Autotrace, or SQL trace with TKPROF, you can get a listing of the total number of data blocks physically read from disk. Also, the `V$SESS_IO` dynamic performance table provides I/O statistics by session. Because you always expect to physically read blocks from the objects in this cache, the number of physical reads for the SQL statement is greater than or equal to the number of blocks read from the object.

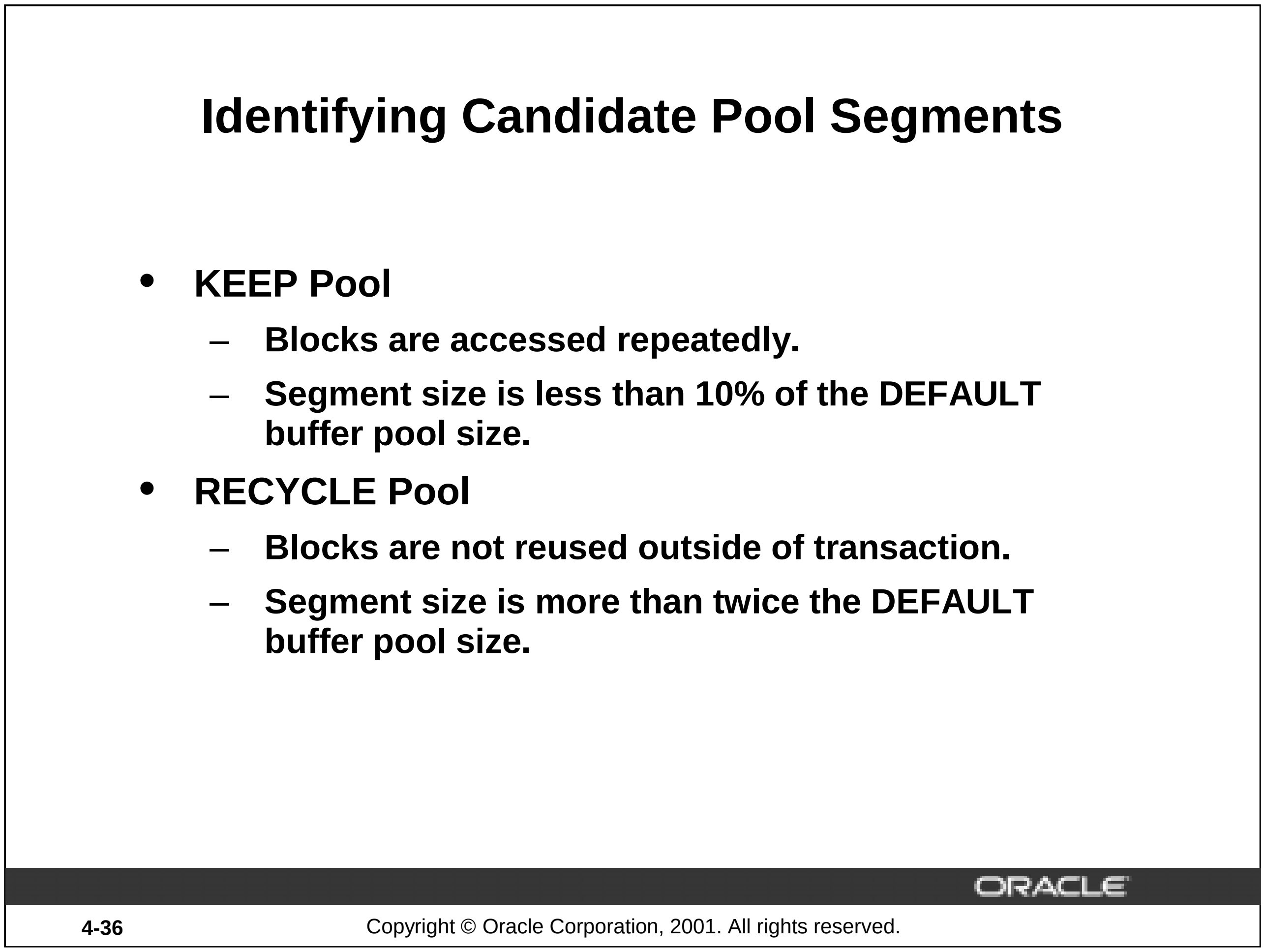
Calculating the Hit Ratio for Multiple Pools

```
SQL>
SQL> SELECT name,
            1 - (physical_reads / (db_block_gets +
                 consistent_gets)) "HIT_RATIO"
 2   FROM   sys.v$buffer_pool_statistics
 3   WHERE  db_block_gets + consistent_gets > 0;

NAME                HIT_RATIO
------------------  ----------
KEEP                .983520845
RECYCLE             .503866235
DEFAULT             .790350047
```

Description of `V$BUFFER_POOL_STATISTICS`

This view displays statistics (physical writes, consistent gets, free buffer waits) against the multiple buffer caches (if allocated):

Column Name	Description
NAME	Name of the buffer pool (KEEP, RECYCLE, DEFAULT)
SET_MSIZE	Maximum buffer size allowed
CNUM_REPL	Current number of buffers in replacement
CNUM_WRITE	Current number of buffers in write list
CNUM_SET	Current total number of buffers in this pool
BUF-GOT	Number of buffers that foreground got for this pool
SUM_WRITE	Number of buffers written by DBW*n* in this pool
SUM_SCAN	Number of buffers scanned by DBW*n* in this pool
FREE_BUFFER_WAIT	Free buffer waits for this pool



Identifying Candidate Pool Segments

- **KEEP Pool**
  - **Blocks are accessed repeatedly.**
  - **Segment size is less than 10% of the DEFAULT buffer pool size.**
- **RECYCLE Pool**
  - **Blocks are not reused outside of transaction.**
  - **Segment size is more than twice the DEFAULT buffer pool size.**

4-36

Trade-Offs

Remember that each object kept in memory results in a trade-off. Although it is beneficial to keep frequently accessed blocks in the cache, retaining infrequently used blocks results in less available space for other more active blocks.

Dictionary Views with Buffer Pools

```
SQL> SELECT  *
  2    FROM  v$buffer_pool;

ID NAME    LO_SETID HI_SETID SET_COUNT BUFFERS LO_BNUM HI_BNUM
-- ------- -------- -------- --------- ------- ------- -------
 1 KEEP           3        3         1   14000       0   13999
 2 RECYCLE        4        6         3    2000   14000   15999
 3 DEFAULT        1        2         2    4000   16000   19999
```

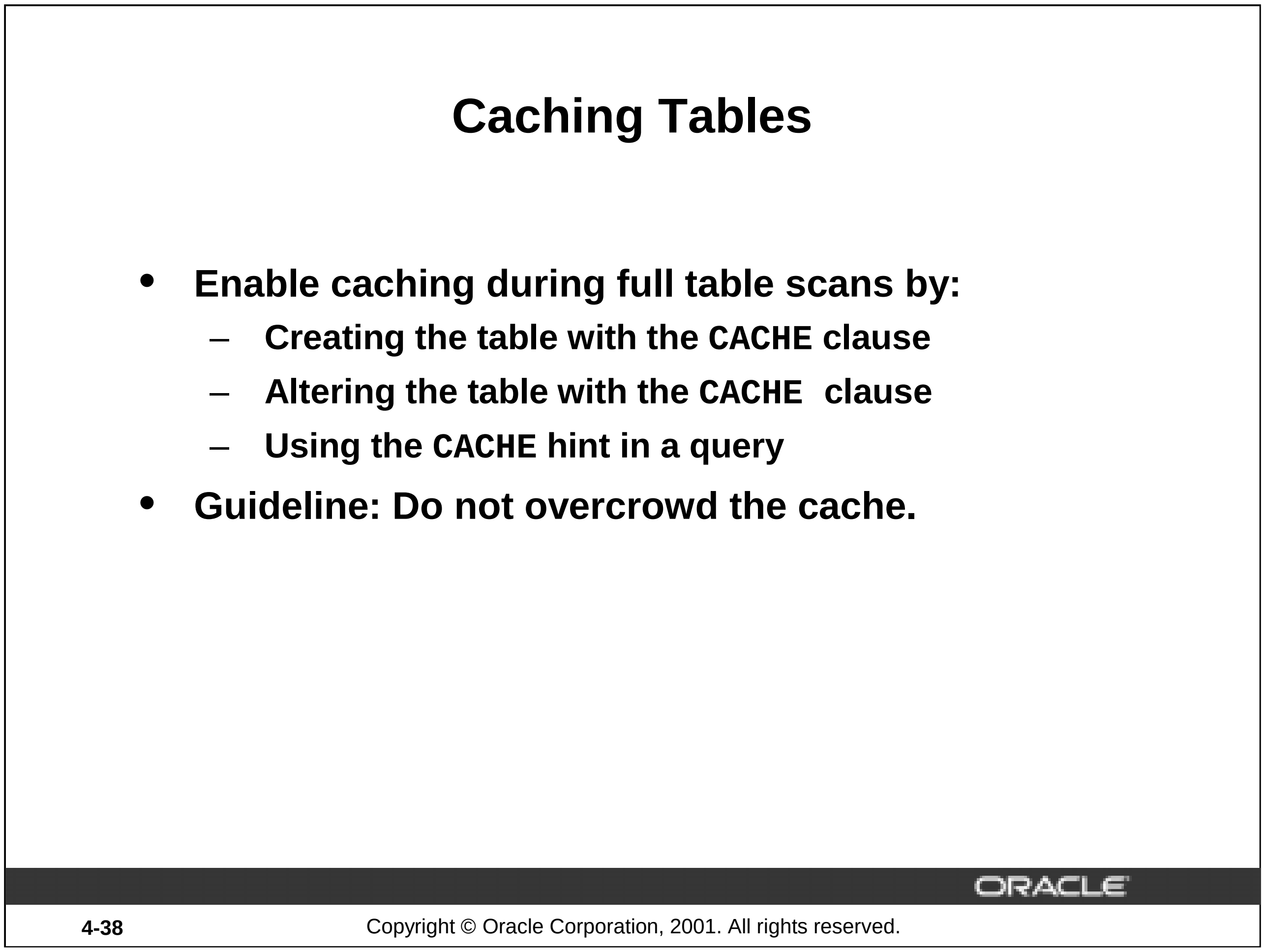
Dictionary Views

These dictionary views have a `BUFFER_POOL` column that indicates the default buffer pool for the given object:

- `USER_SEGMENTS, DBA_SEGMENTS`
- `USER_CLUSTERS, ALL_CLUSTERS, DBA_CLUSTERS`
- `USER_INDEXES, ALL_INDEXES, DBA_INDEXES`
- `USER_TABLES, ALL_TABLES, DBA_TABLES`
- `USER_OBJECT_TABLES, ALL_OBJECT_TABLES, DBA_OBJECT_TABLES`
- `USER_ALL_TABLES, ALL_ALL_TABLES, DBA_ALL_TABLES`

The `V$BUFFER_POOL` view describes the buffer pools allocated. The columns from `V$BUFFER_POOL` show:

- The number and range of LRU latches allocated to the buffer (sets)
- The number and range of blocks allocated to the buffer



Caching Tables

- **Enable caching during full table scans by:**
  - **Creating the table with the `CACHE` clause**
  - **Altering the table with the `CACHE` clause**
  - **Using the `CACHE` hint in a query**
- **Guideline: Do not overcrowd the cache.**

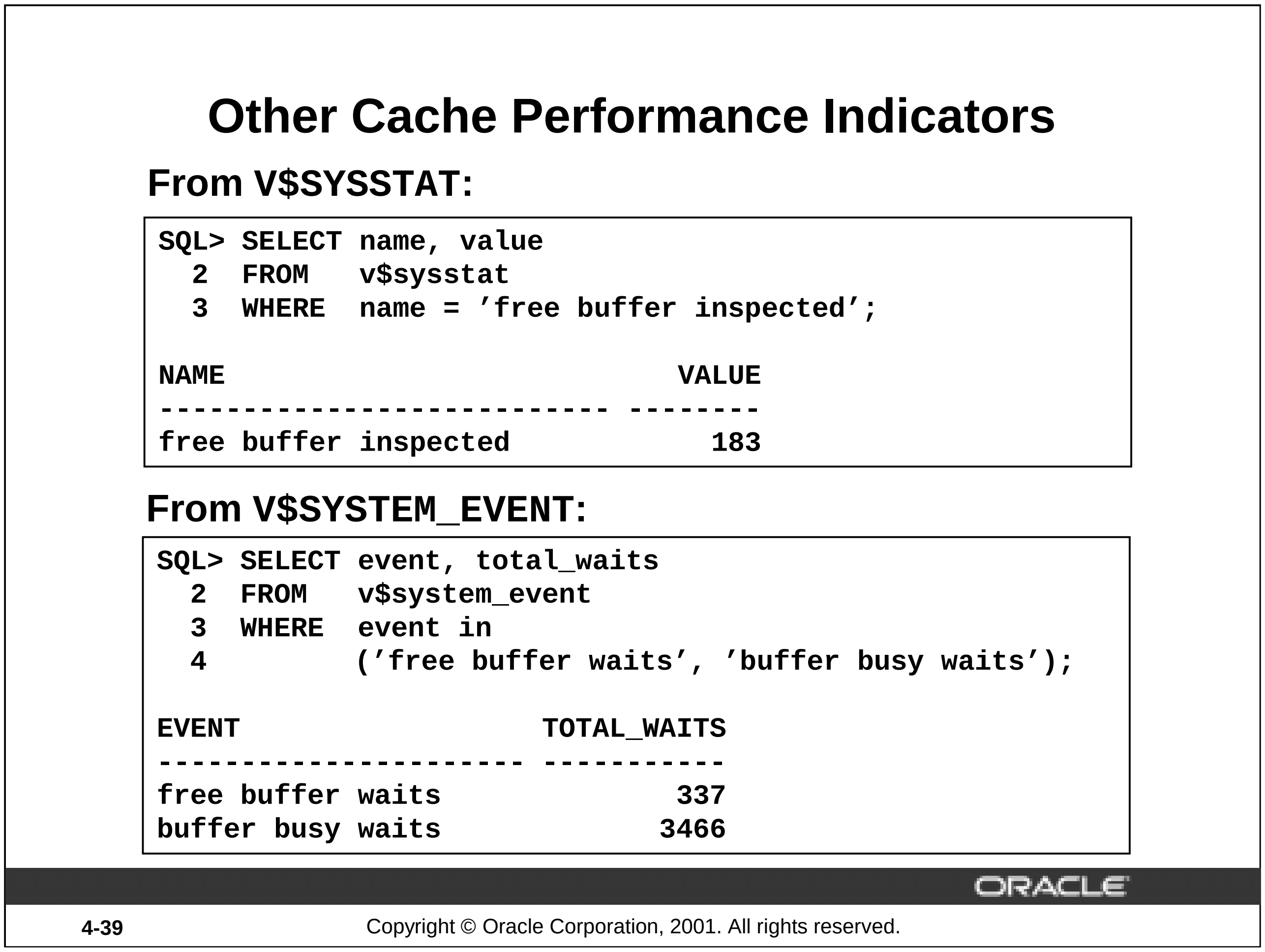
Caching Tables

When the server retrieves blocks using a full table scan, the blocks go to the least recently used end of the LRU list. The blocks are used the next time a free block is needed, so they are not available for other processes. You can choose to cache whole tables at the most recently used (MRU) end of the list.

You alter this behavior if you do the following:

- Create a table using the `CACHE` clause
- Alter a table using the `CACHE` clause
- Code the `CACHE` hint clause into a query

If you use one of these methods, the Oracle server places the table blocks at the most recently used end of the LRU list. Use the `CACHE` clause when you create small lookup tables used by many users. You may overcrowd the buffer cache if you have too many cached tables.



Other Cache Performance Indicators

From V$SYSSTAT:

```
SQL> SELECT name, value
  2  FROM   v$sysstat
  3  WHERE  name = 'free buffer inspected';

NAME                              VALUE
-------------------------- --------
free buffer inspected               183
```

From V$SYSTEM_EVENT:

```
SQL> SELECT event, total_waits
  2  FROM   v$system_event
  3  WHERE  event in
  4         ('free buffer waits', 'buffer busy waits');

EVENT                  TOTAL_WAITS
--------------------- -----------
free buffer waits             337
buffer busy waits            3466
```

4-39

Other Performance Indicators

The buffer cache hit ratio is by far the most useful measure of buffer cache performance. However, there are some other indicators.

Wait Statistics

You should consider increasing the buffer cache size if there are high or increasing values for the Free Buffer Inspected system statistic. This statistic is the number of buffers skipped to find a free buffer. Buffers are skipped because they are dirty or pinned.

Wait Events

You can find out whether there have been waits for buffers from `V$SYSTEM_EVENT` or `V$SESSION_WAIT`. If there are no waits, the event has not yet occurred. There are three main events to look out for:

- Buffer Busy Waits
- This wait indicates that there are some buffers in the buffer cache that multiple processes are attempting to access concurrently. Query `V$WAITSTAT` for the wait statistics for each class of buffer. Common buffer classes that have buffer busy waits include data block, segment header, undo header, and undo block.

Other Performance Indicators (continued)

Buffer Busy Waits (continued)

data block - if the contention is on tables or indexes (not the segment header):

- Check for SQL statements using unselective indexes.
- Check for *right-hand-indexes* (that is, indexes that are inserted at the same point by many processes; for example, those which use sequence number generators for the key values).
- Consider using automatic segment-space management, or increasing free lists to avoid multiple processes attempting to insert into the same block

undo header
Displays contention on rollback segment header: If you are not using automatic undo management, then add more rollback segments.

undo block
Displays contention on rollback segment block: If you are not using automatic undo management, consider making rollback segment sizes larger.

Free Buffer Inspected

This is a measure of how many buffers on the LRU list are inspected by a process looking for a free buffer (writing a new block) before triggering DB writer to flush the dirty buffers to disk.

Free Buffer Waits

This wait event indicates that a server process was unable to find a free buffer and has posted the database writer to make free buffers by writing out dirty buffers. A dirty buffer is a buffer whose contents have been modified. Dirty buffers are freed for reuse when DBWR has written the blocks to disk.

In order to resolve the contention, DBWR has to make blocks available faster for overwriting. To achieve this, examine ways of speeding up the write process. This event is also an indication that the buffer cache is too small. Examine the hit ratios for the buffer cache in order to determine if the cache should be resized.

Causes

DBWR may not be keeping up with writing dirty buffers in the following situations:

- The I/O system is slow.
- The I/O is waiting for resources, such as latches.
- The buffer cache is so small that DBWR spends most of its time cleaning out buffers for server processes.
- The buffer cache is so large that one DBWR process cannot free enough buffers in the cache to satisfy requests.

Actions
If this event occurs frequently, examine the session waits for DBWR to determine whether there is anything delaying DBWR.

Free Lists

- **A free list for an object maintains a list of blocks that are available for inserts.**
- **The number of free lists for an object can be set dynamically.**
- **Single-CPU systems do not benefit greatly from multiple free lists.**
- **The tuning goal is to ensure that an object has sufficient free lists to minimize contention.**
- **Using Automatic Free Space Management eliminates the need for free lists, thus reducing contention on the database.**

Using Free Lists

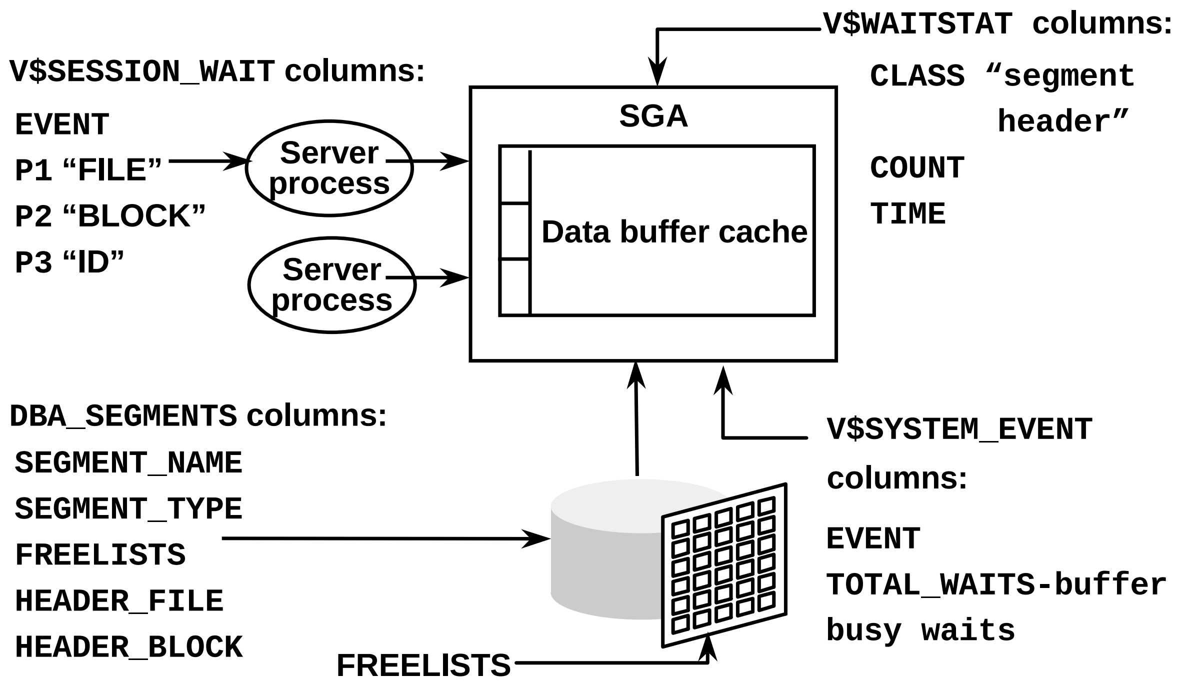
When an insert operation on an object occurs, the free list is used to determine which blocks are available for inserts. Many server processes can contend for the same free list if many inserts are occurring. This results in free list contention while server processes incur waits.

Single-CPU systems do not benefit greatly from multiple free lists, because the CPU manages one process at a time. Even in a single-CPU system, however, adding free lists may ensure that the processor is used more effectively. Care should still be taken when adding free lists.

The overall tuning goal for free lists is to ensure that there is a sufficient number to minimize contention among many server processes.

By using Automatic Free Space Management, Oracle stores the free-list information inside bitmaps that are faster to update, and therefore cause dramatically less contention.

Diagnosing Free List Contention



Dynamic Performance Views

The V$SESSION_WAIT, V$WAITSTAT, and V$SYSTEM_EVENT dynamic performance views are used to diagnose free list contention problems.

The DBA_SEGMENTS data dictionary view is used to identify the objects that need to be modified to increase the number of free lists.

Initialization Parameters

There are no initialization parameters to set for minimizing free list contention. The FREELISTS keyword is used at the segment level. This value cannot be set dynamically; an object must be dropped and re-created in order to change it.

Modifying Free List

The number of free lists in a free list group can be changed by an alter table statement. Note that you cannot alter the free list storage parameter for segments in tablespaces with AUTO SEGMENT SPACE MANAGEMENT.

Diagnostic Criteria

You can query `V$WAITSTAT` and `V$SYSTEM_EVENT` to determine whether there is free list contention. If high numbers are returned, you must identify the object or objects.

- To query `V$WAITSTAT`:

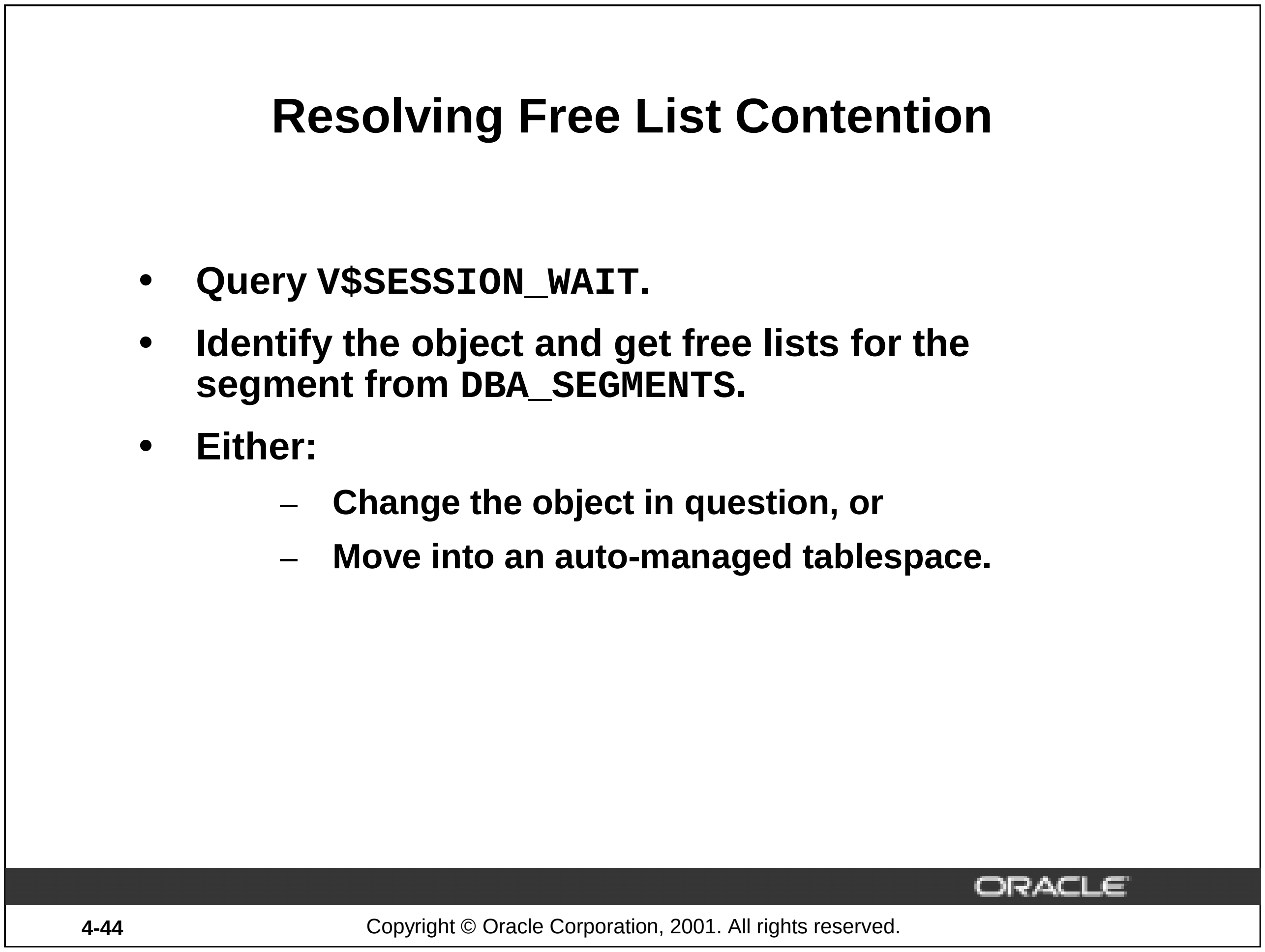
  ```
  SELECT class, count, time
  FROM v$waitstat
  WHERE class = ’segment header’;
  ```

- To query `V$SYSTEM_EVENT`:

  ```
  SELECT event, total_waits
  FROM v$system_event
  WHERE event = ’buffer busy waits’;
  ```

To reduce buffer busy waits on:

- Data blocks: Change `PCTFREE` and/or `PCTUSED`; check for right-hand indexes (indexes that are inserted into at the same point by many processes); increase INITRANS; reduce the number of rows per block
- Segment headers: Use free lists or increase the number of free lists; use free list groups (This can make a difference even in a single instance environment.)
- Free list blocks: Add more free lists (in the case of Oracle Parallel Server, making sure that each instance has its own free list group)



Resolving Free List Contention

- **Query `V$SESSION_WAIT`.**
- **Identify the object and get free lists for the segment from `DBA_SEGMENTS`.**
- **Either:**
  - **Change the object in question, or**
  - **Move into an auto-managed tablespace.**

4-44 Copyright © Oracle Corporation, 2001. All rights reserved.

Identifying the Object

Determine the File, Block, and ID for which free list contention is occurring by querying `V$SESSION_WAIT`.

- Identify the segment and determine the number of free lists that currently exist for the segment identified by querying `DBA_SEGMENTS`:

```
SELECT s.segment_name, s.segment_type, s.freelists,
w.wait_time, w.seconds_in_wait, w.state
FROM dba_segments s, v$session_wait w
WHERE         w.event    ='buffer busy waits‘
AND w.p1 = s.header_file
AND w.p2 = s.header_block;
```

- Use either:
  - Change the object

    To increase the number of free lists for the object, use the `ALTER TABLE` command in Oracle9*i*, specifying the `FREELISTS` keyword.

    or
  - Move the object to an auto managed tablespace. (Discussed later in this chapter)

Auto-management of Free Space

- Create an auto-managed tablespace:

```
CREATE TABLESPACE BIT_SEG_TS
DATAFILE '$HOME/ORADATA/u04/bit_seg01.dbf'
   SIZE 1M
EXTENT MANAGEMENT LOCAL
SEGMENT SPACE MANAGEMENT AUTO;
```

- Create a table that uses auto-management of free space:

```
CREATE TABLE BIT_SEG_TABLE
(IDNUM NUMBER)
TABLESPACE BIT_SEG_TS;
```

Auto-Management of Free Space

With Oracle9*i*, free space can be managed automatically inside database segments. The in-segment free/used space is tracked using bitmaps, as opposed to free lists. Use of bitmaps offers the following benefits:

- Ease of use
- Better space utilization, especially for the objects with highly varying size rows
- Better run-time adjustment to variations in concurrent access
- Better multi-instance behavior in terms of performance/space utilization
- Preparation for future enhancements, such as in-space segment reorganization and in-place tablespace reorganization
- You specify auto-management of free space when you create a tablespace. The specification then applies to all segments subsequently created in this tablespace.

Summary

In this lesson, you should have learned how to:

- Achieve a cache hit ratio greater than or equal to 90%
- Adjust the size of the buffer cache as necessary
- Use the buffer cache advisory feature
- Separate objects into multiple buffer pools
- Use multiple buffer pools
- Cache tables
- Avoid free list contention

Practice 4

The objective of this practice is to use available diagnostic tools to monitor and tune the database buffer cache.

1. Connect as perfstat/perfstat user, and run a statistic snapshot, and note the snapshot number.
2. To simulate user activity against the database, connect as hr/hr user and run the lab04_02.sql script.
3. Connect as system/manager and measure the hit ratio for the database buffer cache using the v$sysstat view. Determine if it is a good ratio or not.
4. Connect as perfstat/perfstat, and run a statistic snapshot, and note the snapshot number. This can be performed by running the script file $HOME/STUDENT/LABS/snap.sql.
5. Use the report from STATSPACK between the last two snapshots to check the buffer cache hit ratio, using the script $HOME/STUDENT/LABS/spreport.sql. Then analyze the buffer hit % in the "Instance Efficiency Percentages" section.

   Note: On a production database if the ratio is bad, add new buffers, run steps 2 to 5, and examine the new ratio to verify that the ratio has improved. If the ratio is good, remove buffers, run steps 2 to 5, and verify if the ratio is still good.
6. Connect as system/manager and determine the size of the table temp_emps in the hr schema that you want to place in the KEEP buffer pool. Do this by using the ANALYZE command, then query the BLOCKS column of the DBA_TABLES view for the table temp_emps.
7. We intend to keep TEMP_EMPS in the KEEP pool. Use the "alter system" command to set DB_KEEP_CACHE_SIZE to 4M for the KEEP pool. This will generate an error due to insufficient memory in the SGA.
8. To resolve the memory problem reduce the size of the shared pool by 8M, using the "alter system" command to set the value of SHARED_POOL_SIZE. Then reissue the command to size the db_keep_cache_size to 4M.

   Note: In a production environment check if you have sufficient SGA size to grow, and if any other component could be reduced in size without adversely affecting performance.
9. Connect as system/manager and enable the TEMP_EMPS table in the hr schema for caching in the keep pool, using the storage clause of the "alter table" command..
10. Connect as hr/hr, and run the script $HOME/STUDENT/LABS/lab04_10.sql. This will execute a query against the temp_emps table in the hr schema.
11. Connect using sys/oracle as sysdba and check for the hit ratio in different buffer pools, using the V$BUFFER_POOL_STATISTICS view.

5

Sizing Other SGA Structures

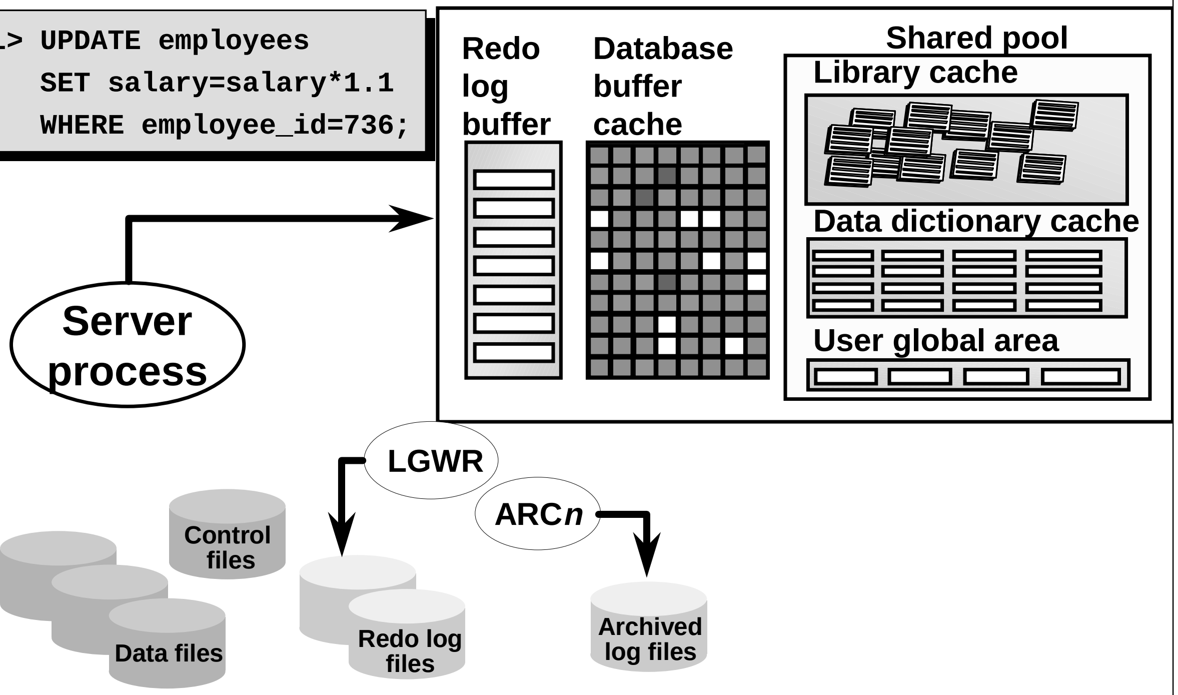
Objectives

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

- Monitor and size the redo log buffer
- Monitor and size the Java pool
- Control the amount of Java session memory used by a session
- Configure the instance to use I/O slaves
- Configure and use multiple DBW processors

The Redo Log Buffer

```
SQL> UPDATE employees
  2   SET salary=salary*1.1
  3   WHERE employee_id=736;
```

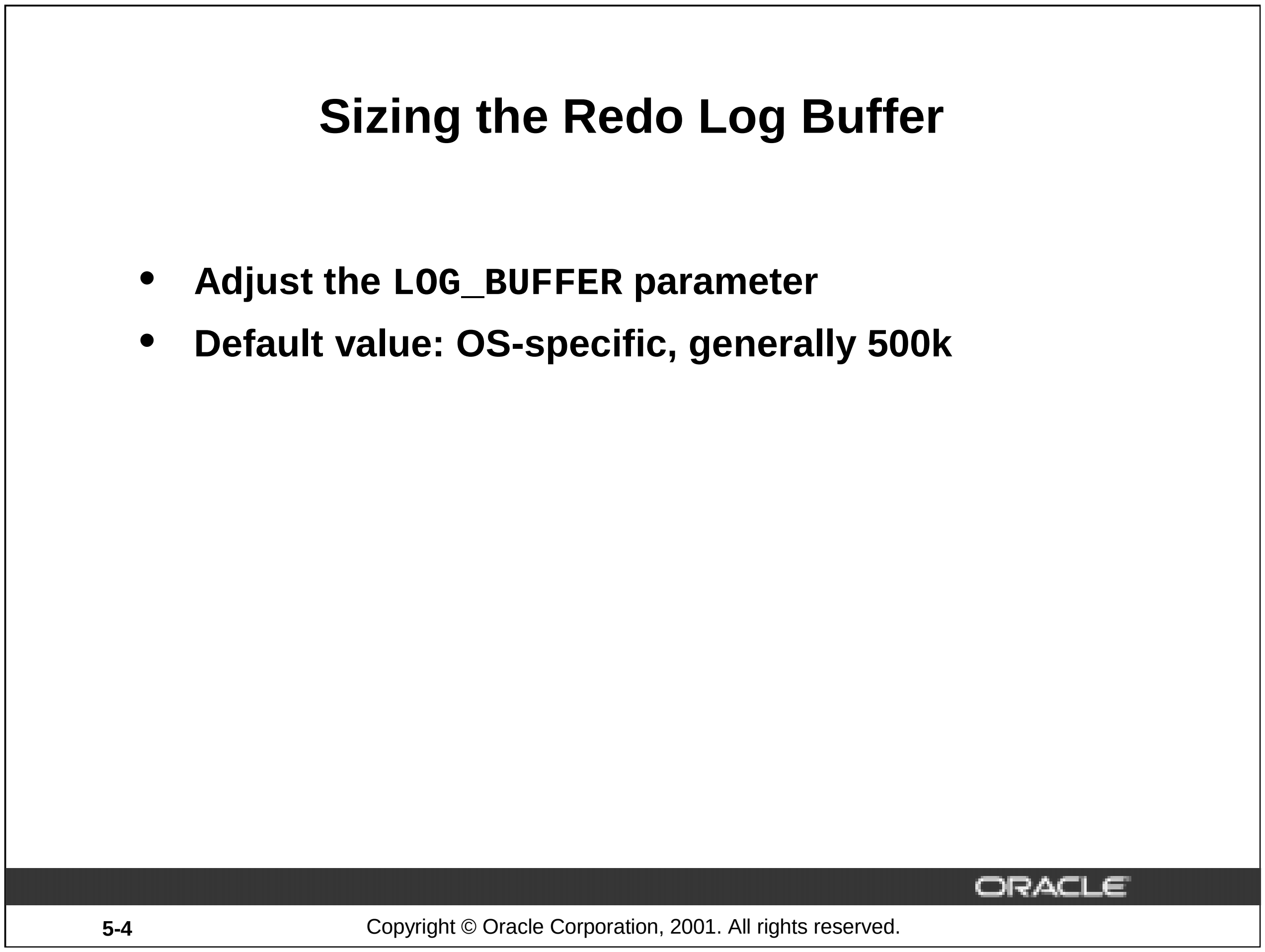


Redo Log Buffer Content

- The Oracle server processes copy redo entries from the user's memory space to the redo log buffer for each DML or DDL statement.
- The redo entries contain the information necessary to reconstruct or redo changes made to the database by `INSERT`, `UPDATE`, `DELETE`, `CREATE`, `ALTER`, or `DROP` operations. They are used for database recovery. They take up continuous, sequential space in the buffer.

Redo Entries and `LGWR`

- The `LGWR` process writes the redo log buffer to the active online redo log file (or members of the active group) on disk. It writes all redo entries that have been copied into the buffer since the last time it wrote.
- The redo log buffer is a circular buffer. Thus, server processes can copy new entries over the entries in the redo log buffer that have been written to disk. `LGWR` normally writes fast enough to ensure that space is always available in the buffer for new entries.


Sizing the Redo Log Buffer

- Adjust the `LOG_BUFFER` parameter
- Default value: OS-specific, generally 500k

5-4 Copyright © Oracle Corporation, 2001. All rights reserved.

Sizing the Redo Log Buffer

- Larger values reduce log file I/O, particularly if transactions are long or numerous.
- Frequent `COMMIT` statements clear out the buffer, leading to a smaller buffer size.
- `Log_buffer` has a minimum size of 64k.

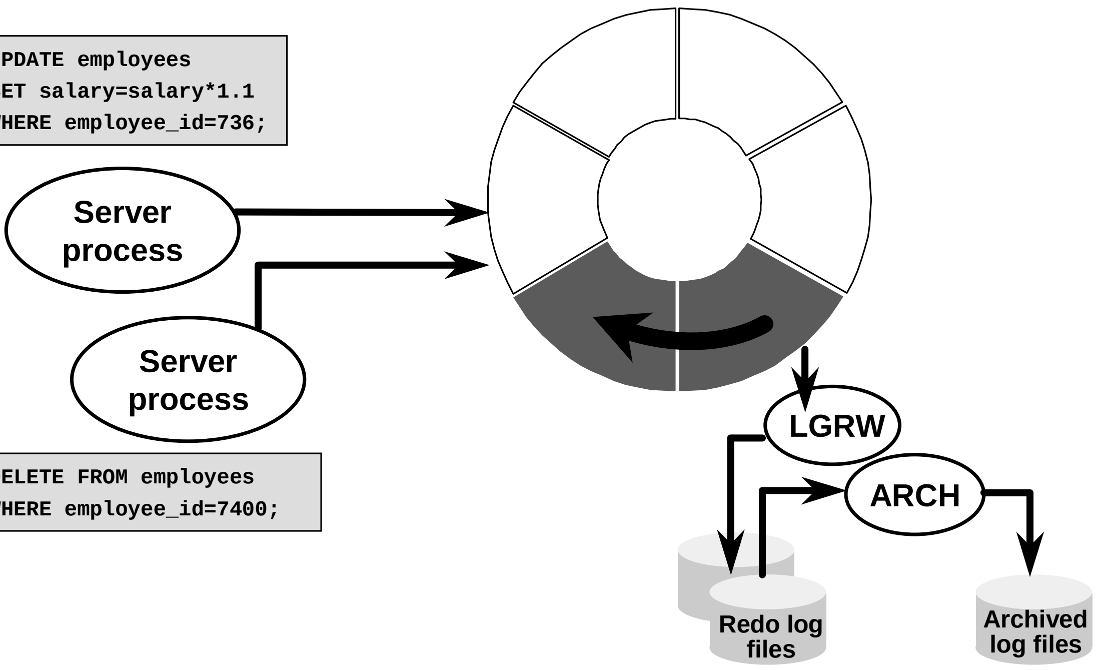
Example

```
 SQL> select 'V$PARAMETER' "Table name", name,
to_number(value,'9999999') "Value"
   2  from v$parameter
   3  where name = 'log_buffer'
   4  UNION
   5  select 'V$SGASTAT' "Table name", name, bytes
   6  from v$sgastat
   7  where name ='log_buffer';

Table name         NAME                 VALUE

--------------     -----------          ---------

V$PARAMETER        log_buffer 120320

V$SGASTAT log_buffer 120320
```

Diagnosing Redo Log Buffer Inefficiency

```
SQL> UPDATE employees
 2   SET salary=salary*1.1
 3   WHERE employee_id=736;
```

```
SQL> DELETE FROM employees
 2   WHERE employee_id=7400;
```



Diagnosing Problems

On machines with fast processors and relatively slow disks, the processors may be filling the rest of the redo log buffer in the time it takes the `LGWR` process to move a portion of the buffer out to disk.

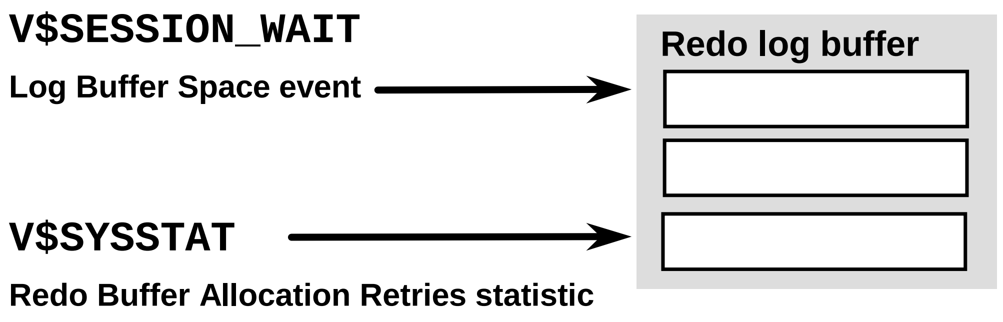
For this reason, a larger buffer makes it less likely that new entries will collide with the part of the buffer still being written.

Server processes may request space from the redo log buffer to write new entries and not find any. They will have to wait for `LGWR` to flush the buffer to disk.

Tuning Goal

Tuning the redo log buffer means ensuring that the space required by server processes in the redo log buffer is sufficient. However, too much space will reduce the amount of memory that can be allocated to other areas. It is also important to note that the DBA can adopt practices that will reduce the amount of redo that must be performed.

Using Dynamic Views to Analyze Redo Log Buffer Efficiency



```
LOG_BUFFER
LOG_CHECKPOINT_INTERVAL
LOG_CHECKPOINT_TIMEOUT
```

Dynamic Views

The `V$SESSION_WAIT` view indicates through the Log Buffer Space event if there are any waits for space in the log buffer because the session is writing data into the log buffer faster than `LGWR` can write it out.

```
SQL> select sid, event, seconds_in_wait, state
   2  from v$session_wait
   3  where event = ’log buffer space%’;

SID    EVENT         SECONDS_IN_WAIT     STATE
-----  ------        ---------------    ---------
log    buffer space  110                 WAITING
```

Redo Buffer Allocation Retries Statistic Ratio

The value of Redo Buffer Allocation Retries should be near 0. This number should not be greater than 1% of the Redo Entries. If this value increments consistently, processes have had to wait for space in the buffer.

Dynamic Views (continued)

The wait may be caused by the log buffer being too small, by checkpointing, or by log switching. In this case you would:

- Increase the size of the redo log buffer, if necessary, by changing the value of the initialization parameter `LOG_BUFFER`.
- Alternatively, improve the checkpointing or archiving process.

The redo log buffer is normally small and a modest increase can greatly enhance throughput.

- The `SECONDS_IN_WAIT` value of the Log Buffer Space event indicates the time spent waiting for space in the redo log buffer because the log switch does not occur.

  This is an indication that the buffers are being filled up faster than `LGWR` is writing. This may also indicate disk I/O contention on the redo log files.
- The Redo Buffer Allocation Retries statistic in `V$SYSSTAT` view reflects the number of times a user process waits for space in the redo log buffer to copy new entries over the entries that have been written to disk. `LGWR` normally writes fast enough to ensure that space is always available in the buffer for new entries, even when access to the redo log is heavy.

```
SQL> SELECT name, value
2   FROM   v$sysstat
3   WHERE  name = 'redo buffer allocation retries';
```

Note: The `V$SYSSTAT` view displays another statistic, Redo Log Space Requests:

```
SQL> select name, value
2  from   v$sysstat
3  where  name='redo log space requests';
```

This statistic indicates that the active log file is full and that the Oracle server is waiting for disk space to be allocated for the redo log entries. Space is made available by performing a log switch.

Redo Log Buffer Tuning Guidelines

- **There should be no Log Buffer Space waits.**

```
SQL> SELECT sid, event, seconds_in_wait, state
 2   FROM   v$session_wait
 3   WHERE  event = ‘log buffer space’;
```

- **The Redo Buffer Allocation Retries value should be near 0; the number should be less than 1% of redo entries.**

```
SQL> SELECT name, value
 2   FROM   v$sysstat
 3   WHERE  name IN (‘redo buffer allocation retries’,
 4                   ‘redo entries’);
```

`SECONDS_IN_WAIT` for Log Buffer Space Event

In `V$SESSION_WAIT`, if the `SECONDS_IN_WAIT` value for the Log Buffer Space event indicates some time spent waiting for space in the redo log buffer, consider:

- Making the log buffer bigger if it is small
- Moving the log files to faster disks such as striped disks

```
SQL> select sid, event, seconds_in_wait, state
   2  from v$session_wait
   3  where event = ’log buffer space%’;
  SID   EVENT      SECONDS_IN_WAIT      STATE
-----   ------     ---------------      -------------
5 log   buffer space         110        WAITING
```

Further Investigations

Investigate the possible reasons why the LGWR is slow in freeing buffers:

- There is disk I/O contention on the redo log files. Check that the redo log files are stored on separate, fast devices.
  - In the V$SYSTEM_EVENT view, check the number of occurrences of the event Log File Switch Completion, which identifies the log file switch waits because of log switches.

    ```
    SQL> select event, total_waits, time_waited,
    average_wait
    2  from v$system_event
    3  where event like 'log file switch completion%';
    ```

  - Increase the size of the redo log files.
- DBW*n* has not completed checkpointing the file when the LGWR needs the file again. LGWR has to wait.
  - In the alert.log file, check for the message "CHECKPOINT NOT COMPLETE."
  - In the V$SYSTEM_EVENT view, check the number of occurrences of the event Log File Switch (Checkpoint Incomplete), which identifies the log file switch waits because of incomplete checkpoints.

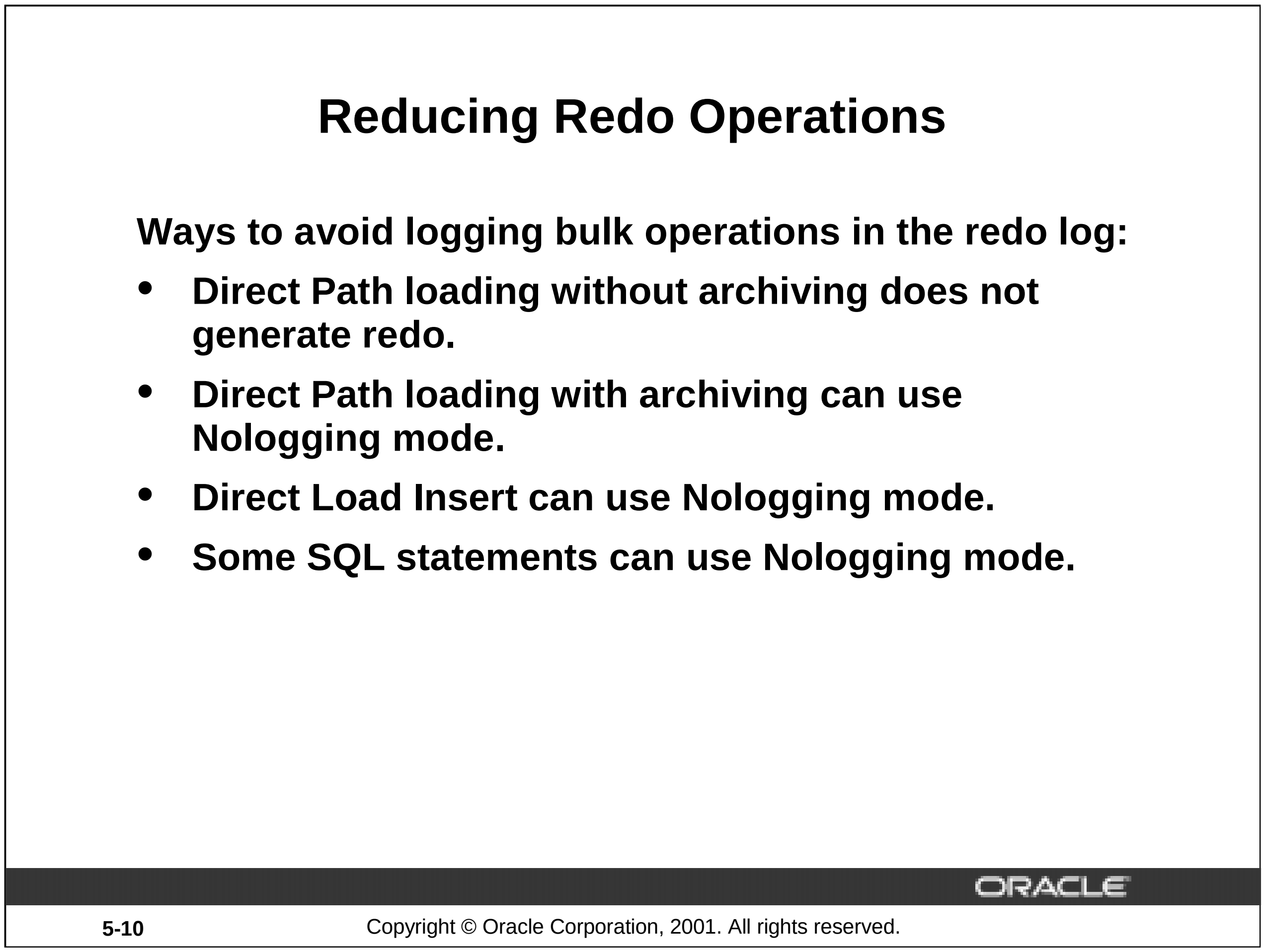
    ```
    SQL>select event, total_waits, time_waited, average_wait
    2  from v$system_event
    3  where event like 'log file switch (check%';
    ```

  - Check the frequency of checkpoints and set the appropriate values for LOG_CHECKPOINT_INTERVAL and LOG_CHECKPOINT_TIMEOUT.
  - Check the size and number of redo log groups.
- The archiver cannot write to the archived redo log files or cannot achieve the archive operation fast enough. Therefore, it prevents the LGWR from writing.
  - Confirm that the archive device is not full and add redo log groups.
  - In the V$SYSTEM_EVENT view, check the number of the occurrences of the event Log File Switch (Archiving Needed), which identifies the log file switch waits because of the archiving issue.

    ```
    SQL> select event, total_waits, time_waited,
    average_wait
    2  from v$system_event
    3  where event like 'log file switch (arch%';
    ```

  - Regulate archiving speed.

  The LGWR process starts a new ARC*n* process whenever the current number of ARC*n* processes is insufficient to handle the workload. If you anticipate a heavy workload for archiving, such as during bulk loading of data, specify multiple archiver processes with the LOG_ARCHIVE_MAX_PROCESSES initialization parameter.
- DB_BLOCK_CHECKSUM is set to TRUE, and therefore adds performance overhead.



Reducing Redo Operations

Ways to avoid logging bulk operations in the redo log:

- **Direct Path loading without archiving does not generate redo.**
- **Direct Path loading with archiving can use Nologging mode.**
- **Direct Load Insert can use Nologging mode.**
- **Some SQL statements can use Nologging mode.**

5-10

SQL*Loader and the Nologging Mode

Conventional path loading generates redo log entries just as any DML statement. When using direct path, 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.)

Direct Load Insert and Nologging Mode

The `Nologging` option:

- Applies to tables, tablespaces, and indexes
- Does not record changes to data in the redo log buffer (Some minimal logging is still carried out, for operations such as extent allocation.)
- Is not specified as an attribute at the `INSERT` statement level, but is instead specified when using the `ALTER` or `CREATE` command for the table, index, or tablespace

Direct Load Insert and Nologging Mode (continued)

- The mode is set before the load and is reset to LOGGING once the load completes (If a media failure occurs before a backup is taken, then all tables, and indexes that have been modified, may be corrupted.)

SQL Statements that Can Use Nologging Mode

Although you can set the `NOLOGGING` attribute for a table, index, or tablespace, Nologging mode does not apply to every operation on the object for which the attribute is set.

- `CREATE TABLE ... AS SELECT`
- `CREATE INDEX`
- `ALTER INDEX ... REBUILD`

The following statements are nevertheless unaffected by the `NOLOGGING` attribute: `UPDATE`, `DELETE`, conventional path `INSERT`, and various DDL statements not listed above.

Note: For backward compatibility, `UNRECOVERABLE` is still supported as an alternate keyword with the `CREATE TABLE` statement in Oracle8*i* Server release 8.1. This alternate keyword may not be supported in future releases.

Monitoring Java Pool Memory

```
SQL> SELECT * FROM v$sgastat
  2  WHERE pool = 'java pool';
POOL        NAME                       BYTES
----------- -------------------- ----------
java pool   free memory             30261248
java pool   memory in use           19742720
```

Limit Java session memory usage:

- **`JAVA_SOFT_SESSIONSPACE_LIMIT`**
- **`JAVA_MAX_SESSIONSPACE_SIZE`**

5-12 Copyright © Oracle Corporation, 2001. All rights reserved.

Limiting Java Session Memory

These parameters allow the DBA to limit the amount of memory used for each Java session, and are discussed later in this lesson.

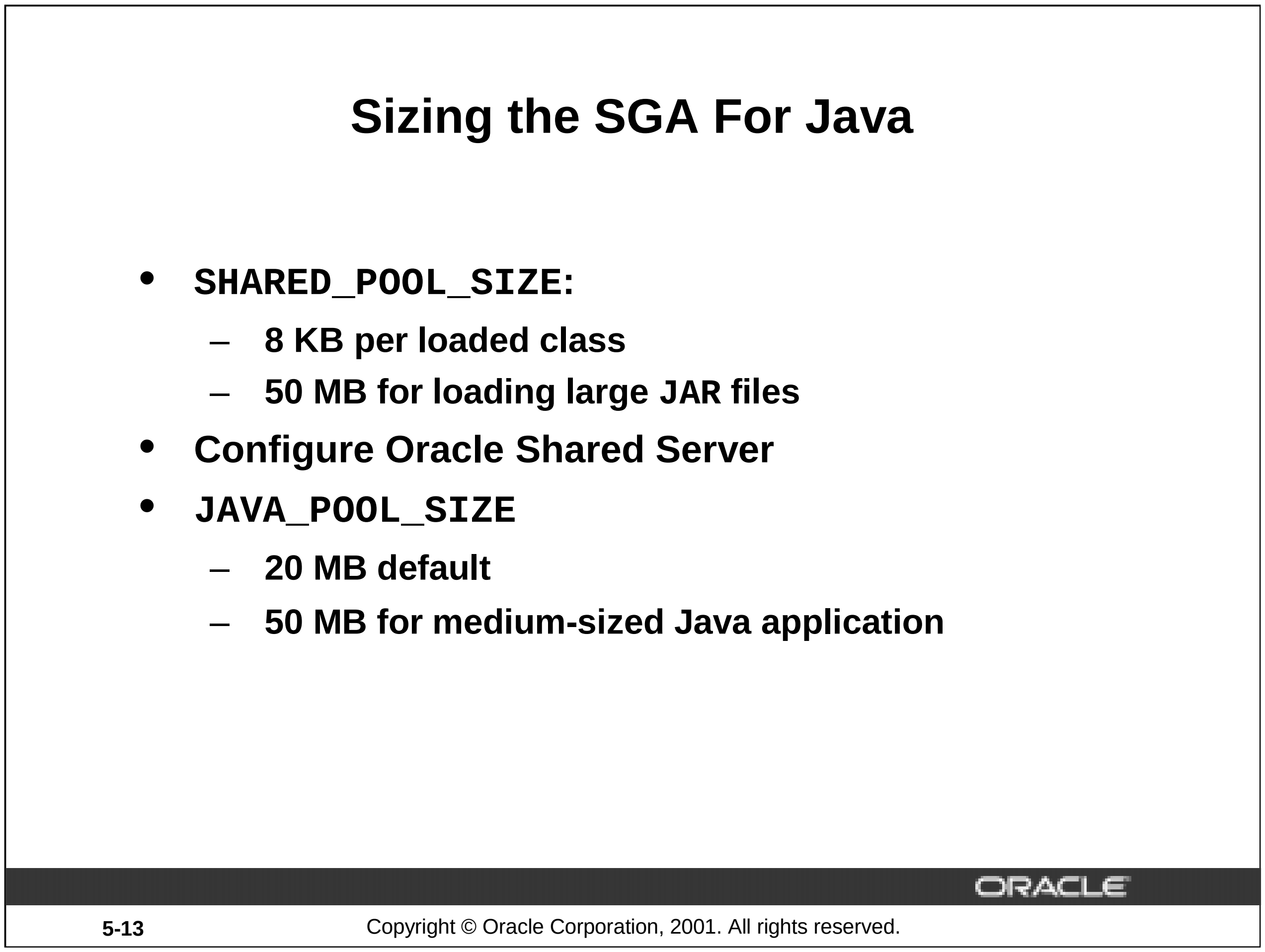
`JAVA_SOFT_SESSIONSPACE_LIMIT`

When a user's session-duration Java state exceeds this size, a warning is written into an RDBMS trace file. The default is 1M. This parameter allows you to specify a soft limit on Java memory usage in a session, as a means to warn you if something is awry.

`JAVA_MAX_SESSIONSPACE_SIZE`

When a user's session-duration Java state attempts to exceed this size, the session is killed with an out-of-memory failure. The default is 4G. This limit is purposely set very high so as not to be visible normally. If a user-invoked Java program is not self-limiting in its memory usage, this setting can place a hard limit on the amount of session space made available to it. When the value for this parameter is exceeded, Oracle9*i* displays the following error message:

```
ORA-29554: unhandled Java out of memory condition
```



Sizing the SGA For Java

- `SHARED_POOL_SIZE`:
  - 8 KB per loaded class
  - 50 MB for loading large JAR files
- Configure Oracle Shared Server
- `JAVA_POOL_SIZE`
  - 20 MB default
  - 50 MB for medium-sized Java application

5-13

How Oracle9*i* Enterprise Java Engine (EJE) Uses the Shared and Large Pool

The Java Engine uses about 8 KB per loaded class. The shared pool is also temporarily used by the class loader while loading and resolving classes into the database. While loading and resolving particularly large JAR files, you can use 50 MB of shared pool memory.

The UGA is allocated in the large pool when the `LARGE_POOL_SIZE` is included in the `init.ora` file.

How Oracle9*i* Enterprise Java Engine (EJE) Uses the Java Pool

The Java pool is a structure in the SGA that is used for all session-specific Java code and data within the EJE. During instance startup, the Java pool is allocated a fixed amount of memory equal to the `init.ora` parameter `JAVA_POOL_SIZE`.

Generally, the `JAVA_POOL_SIZE` should be set to 50 MB or higher for large applications. The default value of 20 MB should be adequate for typical Java Stored Procedure usage.

Multiple I/O Slaves

- **Provide nonblocking asynchronous I/O requests**
- **Are deployed by the `DBW0` process**
- **Are typically not recommended if asynchronous I/O is available**
- **Follow the naming convention `ora_i`*`nnn`*`_SID`**
- **Turn asynchronous I/O on or off with:**
  - **`DISK_ASYNCH_IO`**
  - **`TAPE_ASYNCH_IO`**

I/O Slaves

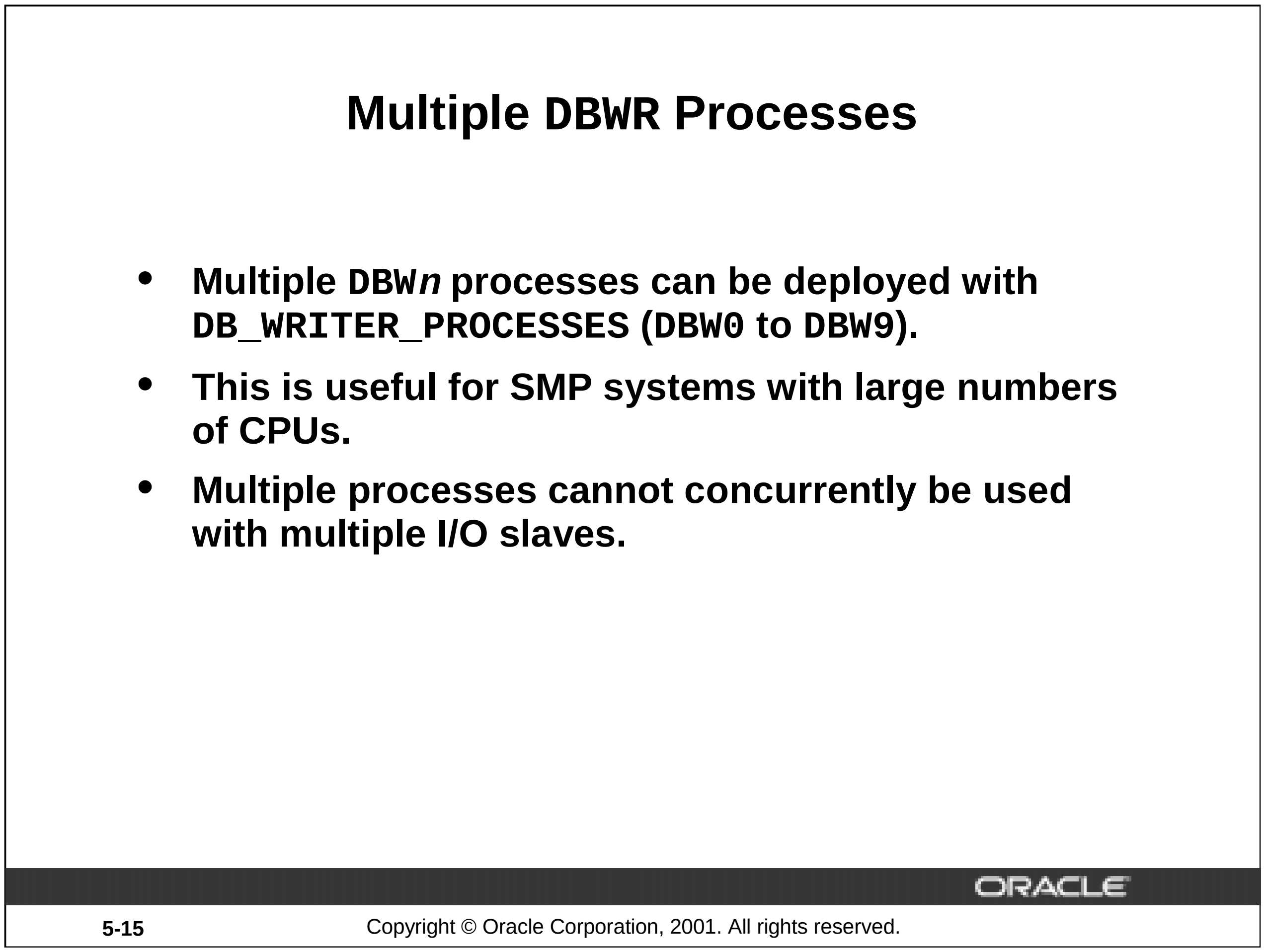
Asynchronous I/O behavior, if not natively available, can be simulated with the deployment of I/O slave processes. I/O slaves are specialized processes whose only function is to perform I/O. Many platforms that support asynchronous I/O for disk devices do not do so for tape devices. In this case, I/O slaves can be used to do nonblocking I/O to tape devices.

The `DBWR_IO_SLAVES` and `BACKUP_TAPE_IO_SLAVES` initialization parameters control I/O slave deployment. You can turn asynchronous I/O on and off with the `DISK_ASYNCH_IO` and `TAPE_ASYNCH_IO` parameters. It may be necessary to turn off the asynchronous I/O facility provided by the operating system. For example, if the asynchronous I/O code of the platform has bugs or is not efficient, asynchronous I/O can be disabled by device type. Usually the parameter should be left at the default value of TRUE.

The I/O Slave Mechanism

I/O slaves can be deployed by the `DBW0` process. I/O slaves for `DBW0` are allocated immediately following database open when the first I/O request is made.

The DBW0 process looks for an idle I/O slave. If one is available it gets the post. If there are no idle slaves, the I/O issuer spawns one. If the allowed number of slaves have been spawned, the issuer waits and tries again to find an idle slave. The `DBW0` continues to do all the `DBW0`-related work; for example, gathering dirty buffers into a batch. The `DBW0` I/O slave simply does the I/O on behalf of `DBW0.` That is, the writing of the batch is parallelized between the I/O slaves. This is beneficial in write-intensive environments, because the CPU time associated with issuing I/Os can be divided between the I/O slaves.



Multiple `DBWR` Processes

- **Multiple `DBW`*`n`* processes can be deployed with `DB_WRITER_PROCESSES` (`DBW0` to `DBW9`).**
- **This is useful for SMP systems with large numbers of CPUs.**
- **Multiple processes cannot concurrently be used with multiple I/O slaves.**

5-15

The Multiple `DBWR`s Mechanism

Multiple `DBWR` processes can be specified by the `DB_WRITER_PROCESSES` parameter. Up to ten processes (`DBW0` to `DBW9`) can be used. In contrast to the multiple I/O slaves, which only parallelize the writing of the batch between the `DBWR` I/O slaves, you can parallelize the gathering as well as the writing of buffers with the multiple `DBWR` feature.

Therefore, *n* `DBWR` processes should deliver more than the throughput of one `DBW0` process with the same number *n* of I/O slaves.

Tuning DBW*n* I/O

Tune the DBW*n* by looking at the value of the following event:

```
FREE BUFFER WAITS
```

Guidelines

Consider increasing the DBW*n* processes, if you see a high number of `free_buffer_waits` after querying the `V$SYSTEM_EVENT` view as in the following syntax:

```
SELECT TOTAL_WAITS FROM V$SYSTEM_EVENT WHERE EVENT =
  'FREE BUFFER WAITS';
```

Summary

In this lesson, you should have learned how to:

- Monitor and size the redo log buffer
- Monitor and size the Java pool
- Control the amount of Java session memory used by a session
- Configure multiple I/O slaves
- Use multiple DBW processors

Practice 5

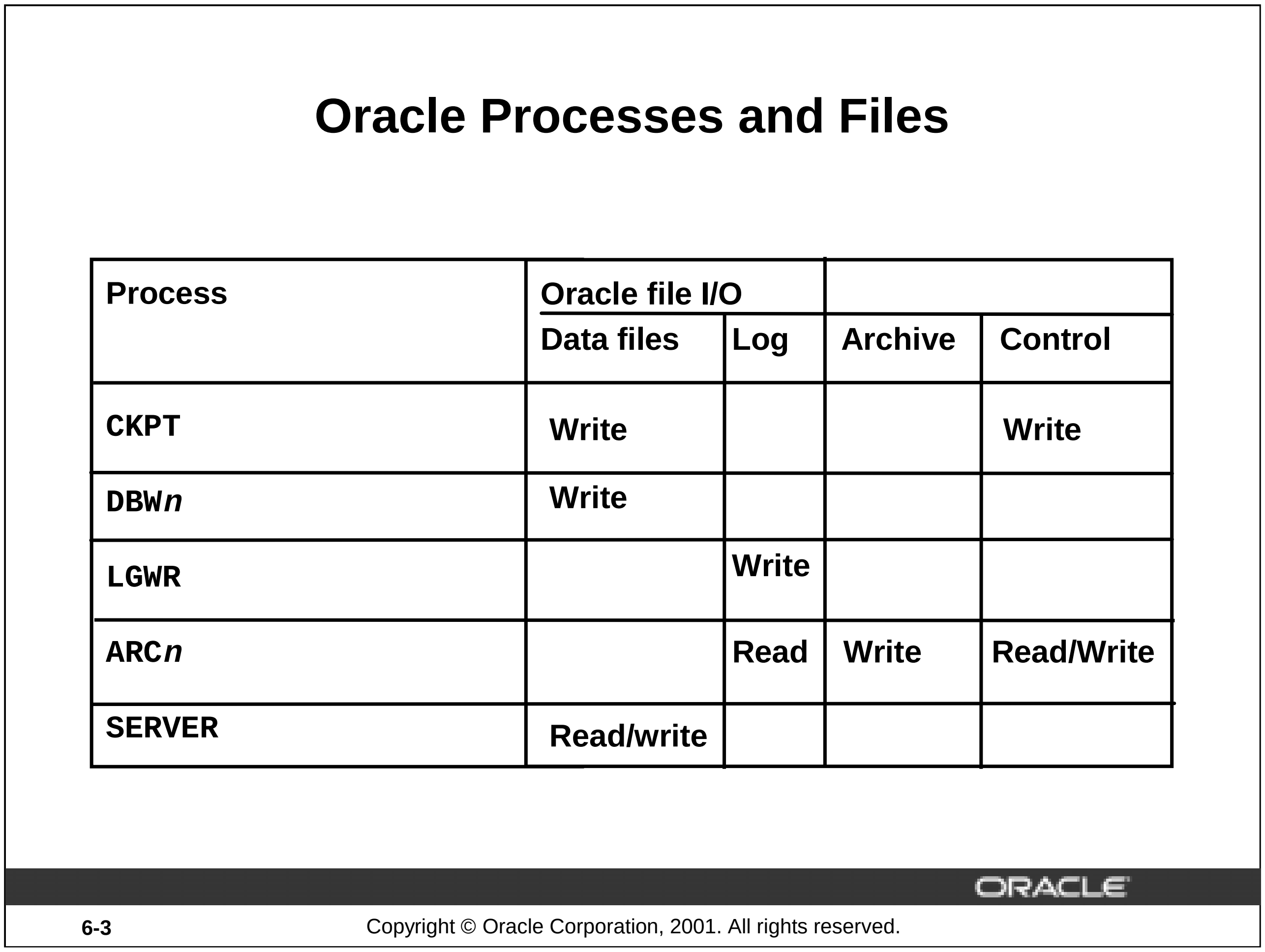
1. Connect as perfstat/perfstat and collect a snapshot of the current statistics by running the script $HOME/STUDENT/LABS/snap.sql. Record the snapshot id for later use.
2. Connect as user SH/SH and run the $HOME/STUDENT/LABS/lab05_02.sql script in the STUDENT/LABS directory in order to have a workload.
3. Connect as system/manager and query the V$SYSSTAT view to determine if there are space requests for the redo log buffer.
4. Connect as perfstat/perfstat and collect another set of statistics using the $HOME/STUDENT/LABS/snap.sql script. Then use $HOME/STUDENT/LABS/spreport.sql to generate a report using the two snapshot ids that you have collected. From the report determine log buffer statistics. View the generated file using an editor, and locate the “log buffer space” statistic.
5. Increase the size of the redo log buffer in the init.ora file located in the $HOME/ADMIN/PFILE directory.

6 Database Configuration and I/O Issues

Objectives

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

- List the advantages of distributing different Oracle file types
- List reasons for partitioning data in tablespaces
- Diagnose tablespace usage problems
- Describe how checkpoints work
- Monitor and tune checkpoints
- Monitor and tune redo logs

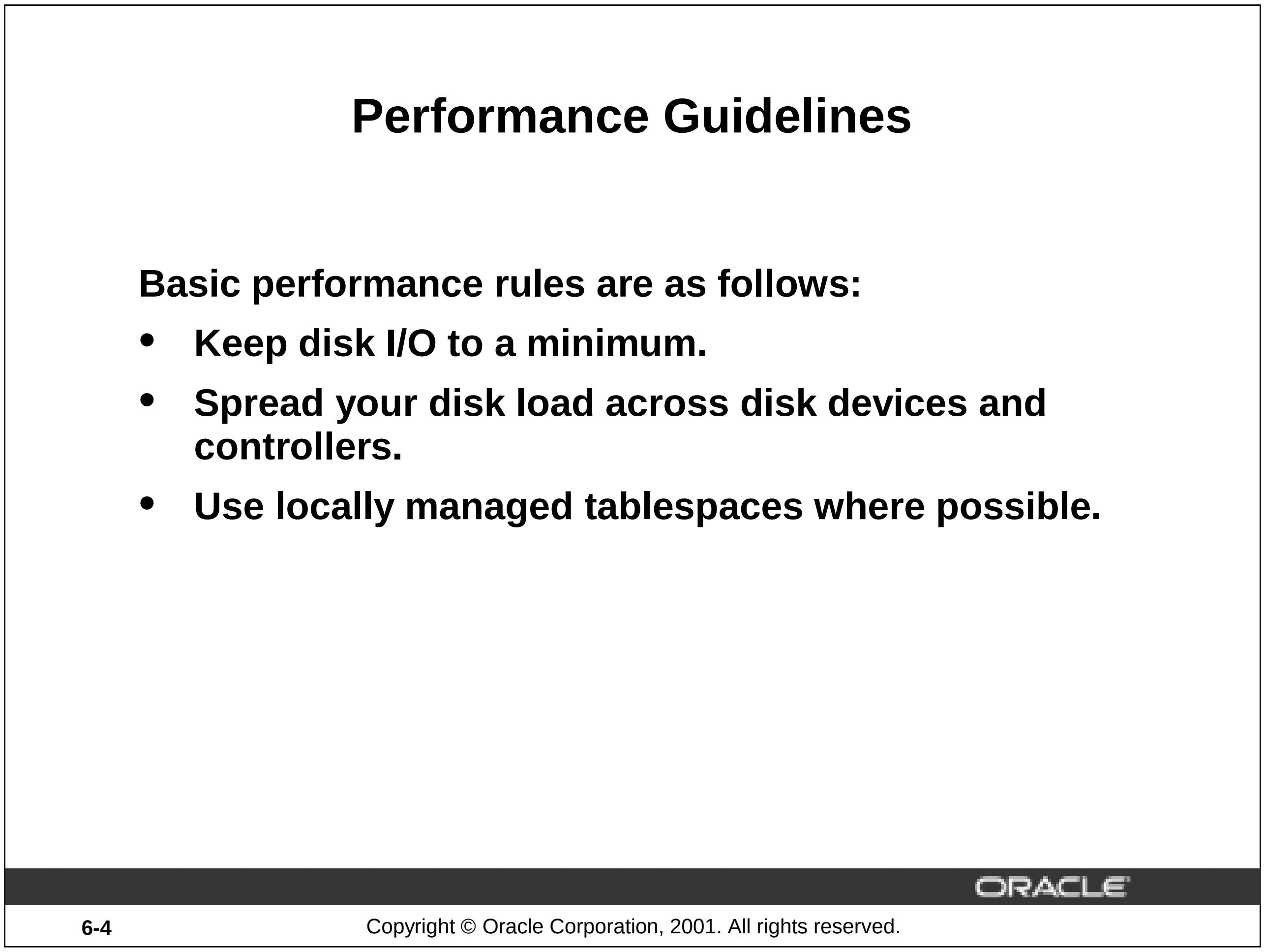


Oracle Processes and Files

Process	Oracle file I/O			
	Data files	Log	Archive	Control
`CKPT`	Write			Write
`DBWn`	Write			
`LGWR`		Write		
`ARCn`		Read	Write	Read/Write
`SERVER`	Read/write			

Oracle Processes

While most server processes only *read* from disk, `sort direct write` server processes will *write* to disk.



Performance Guidelines

Basic performance rules are as follows:

- **Keep disk I/O to a minimum.**
- **Spread your disk load across disk devices and controllers.**
- **Use locally managed tablespaces where possible.**

Performance Guidelines

For very heavy OLTP applications, there are concurrency problems when using dictionary managed tablespaces because the dictionary needs to be accessed for space management operations during extent allocation. With locally managed tablespaces, there is no dictionary intervention, and therefore fewer concurrency problems.

Tables created in locally managed tablespaces can have thousands of extents with no performance implications, therefore reorganizing tables is unnecessary. In locally managed tablespace, free space does not have to be coalesced to remove “beehive” fragmentation because bitmaps track free space and allocate it effectively.

When users are created, a temporary tablespace is designated for any disk sorts that they will need. These sort areas should be separate from other database objects. If users do not have a temporary tablespace, any sort areas needed will use the SYSTEM tablespace.

Tables and indexes should be split into separate tablespaces, because indexes and tables are often inserted into and read from simultaneously.

All tables that contain `LONG` or `LOB` data types, for example, `BLOB` and `CLOB,` should be placed into a separate tablespace.

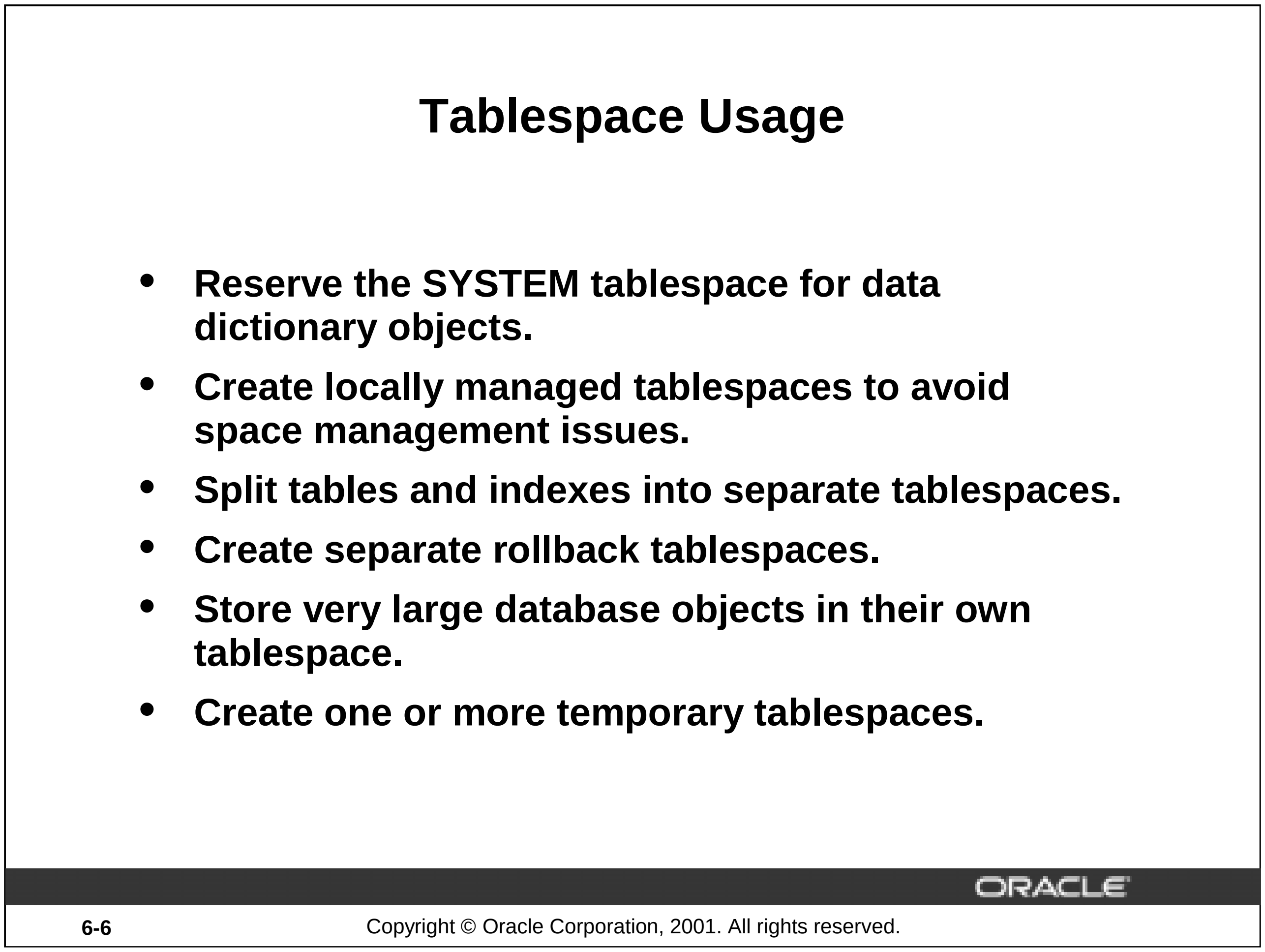
Distributing Files across Devices

- **Separate data files and redo log files.**
- **Stripe table data.**
- **Reduce disk I/O unrelated to the server.**
- **Evaluate the use of raw devices.**

6-5

Guidelines

- In general, to reduce the activity on an overloaded disk, move one or more of its heavily accessed files to a less active disk.
  - Redo log files are written sequentially by the `LGWR` process. Put the redo log files on a disk with no other activity or a low incidence of reads and writes: `LGWR` can write much faster if there is no concurrent activity.
  - If users concurrently access a large table, striping across separate data files and disks can help to reduce contention.
- Try to eliminate I/O unrelated to the Oracle server on disks that contain database files. This is also helpful in optimizing access to redo log files and enables you to monitor all data file activities on such disks with the `V$FILESTAT` dynamic performance view.
- Knowing the types of operation that predominate in your application and the speed with which your system can process the corresponding I/Os, you can choose the disk layout that maximizes performance.



Tablespace Usage

- **Reserve the SYSTEM tablespace for data dictionary objects.**
- **Create locally managed tablespaces to avoid space management issues.**
- **Split tables and indexes into separate tablespaces.**
- **Create separate rollback tablespaces.**
- **Store very large database objects in their own tablespace.**
- **Create one or more temporary tablespaces.**

6-6

Guidelines

Each database should have separate tablespaces specified for:

- Data dictionary objects
- Rollback segments and undo segments
- Temporary segments
- Tables
- Indexes
- Very large objects

Most production databases have many more tablespaces than this, but the important principle is to separate data of different types and with different uses for housekeeping and backup purposes.

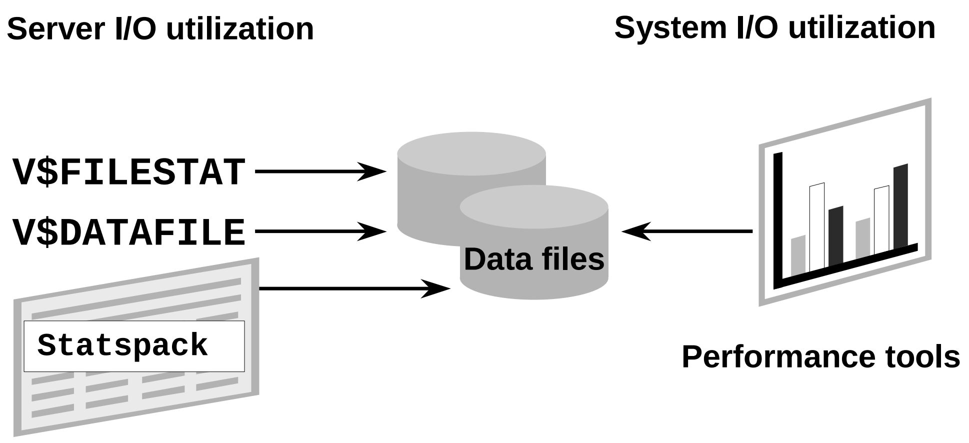
The SYSTEM tablespace contains only data dictionary objects owned by `sys.` No other users should have the ability to create objects in it.

Remember that stored objects such as packages and database triggers form part of the data dictionary.

Rollback segments should use rollback segment tablespaces exclusively.

Undo segments can only exist in UNDO tablespaces thus making them exclusive.

Diagnostic Tools for Checking I/O Statistics



Monitoring Use of Files

To monitor which files are subject to most I/O in an existing database, you can query the following:

- `V$FILESTAT` view
- File I/O monitor using Enterprise Manager
- File statistics in `STATSPACK`

Using the V$FILESTAT Dynamic Performance View

You can query V$FILESTAT to find out the number of disk I/Os per disk file.

Summarize all of the I/Os on data files on a per-disk basis to find the data files most likely to cause a disk bottleneck.

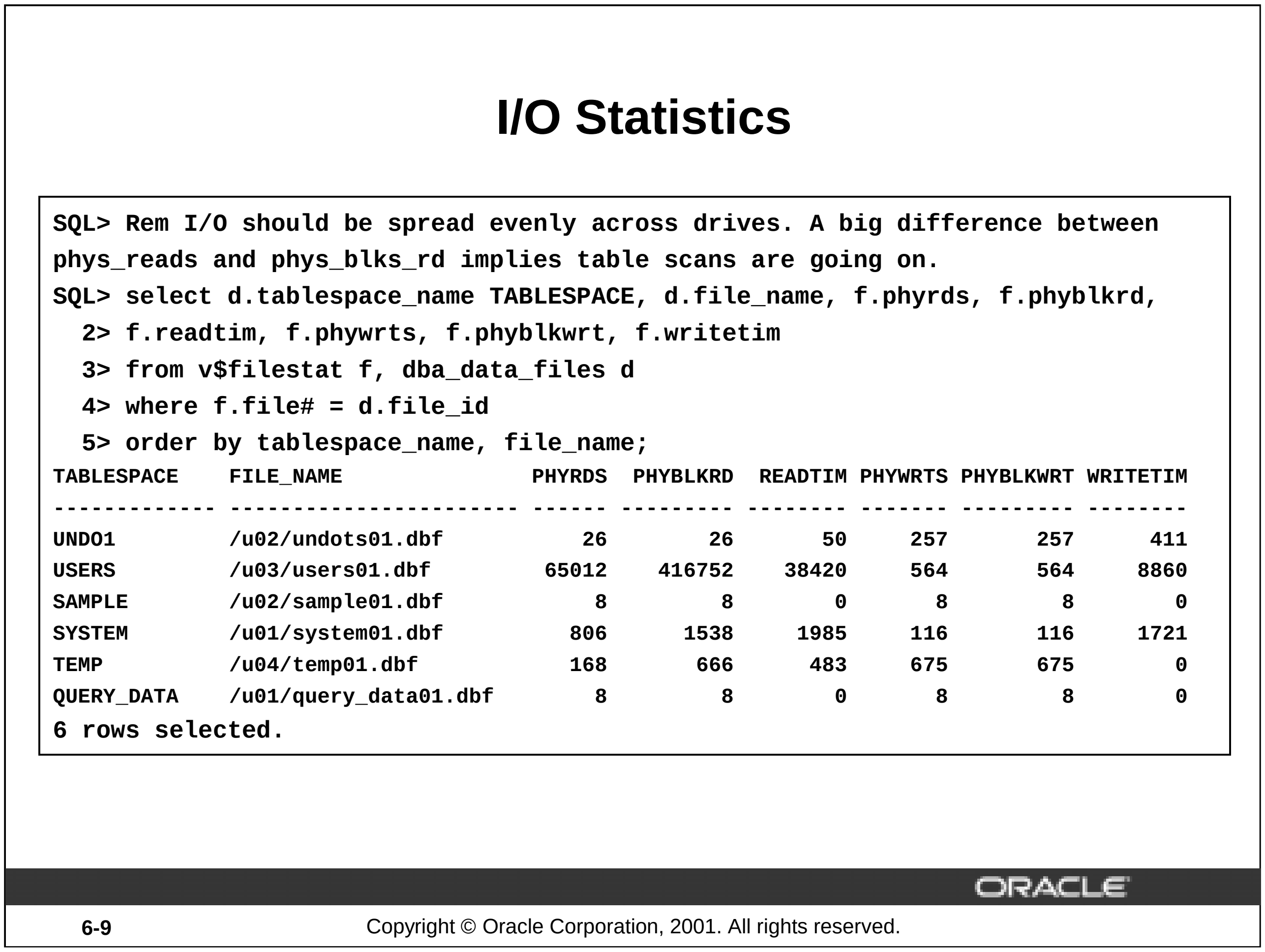
V$FILESTAT contains the following columns:

Context	Reference
file#	File number (join to file# in V$DATAFILE for the name)
Phyrds	Number of physical reads done
phywrts	Number of physical writes done
phyblkrd	Number of physical blocks read
phyblkwrt	Number of physical blocks written
readtim	Time spent doing reads
writetim	Time spent doing writes

Note: The last two columns contain 0 unless the TIMED_STATISTICS parameter is set to TRUE.

Use the following query to monitor these values:

```
SQL> SELECT phyrds,phywrts,d.name
  2  FROM v$datafile d, v$filestat f
  3  WHERE d.file#=f.file# order by d.name;
    PHYRDS  PHYWRTS    NAME
---------- --------    --------------------------------
       806      116    /…/u01/system01.dbf
       168      675    /…/u04/temp01.dbf
         8        8    /…/u02/sample01.dbf
        26      257    /…/u02/undots01.dbf
     65012      564    /…/u03/users01.dbf
         8        8    /…/u01/query_data01.dbf
  6 rows selected
```

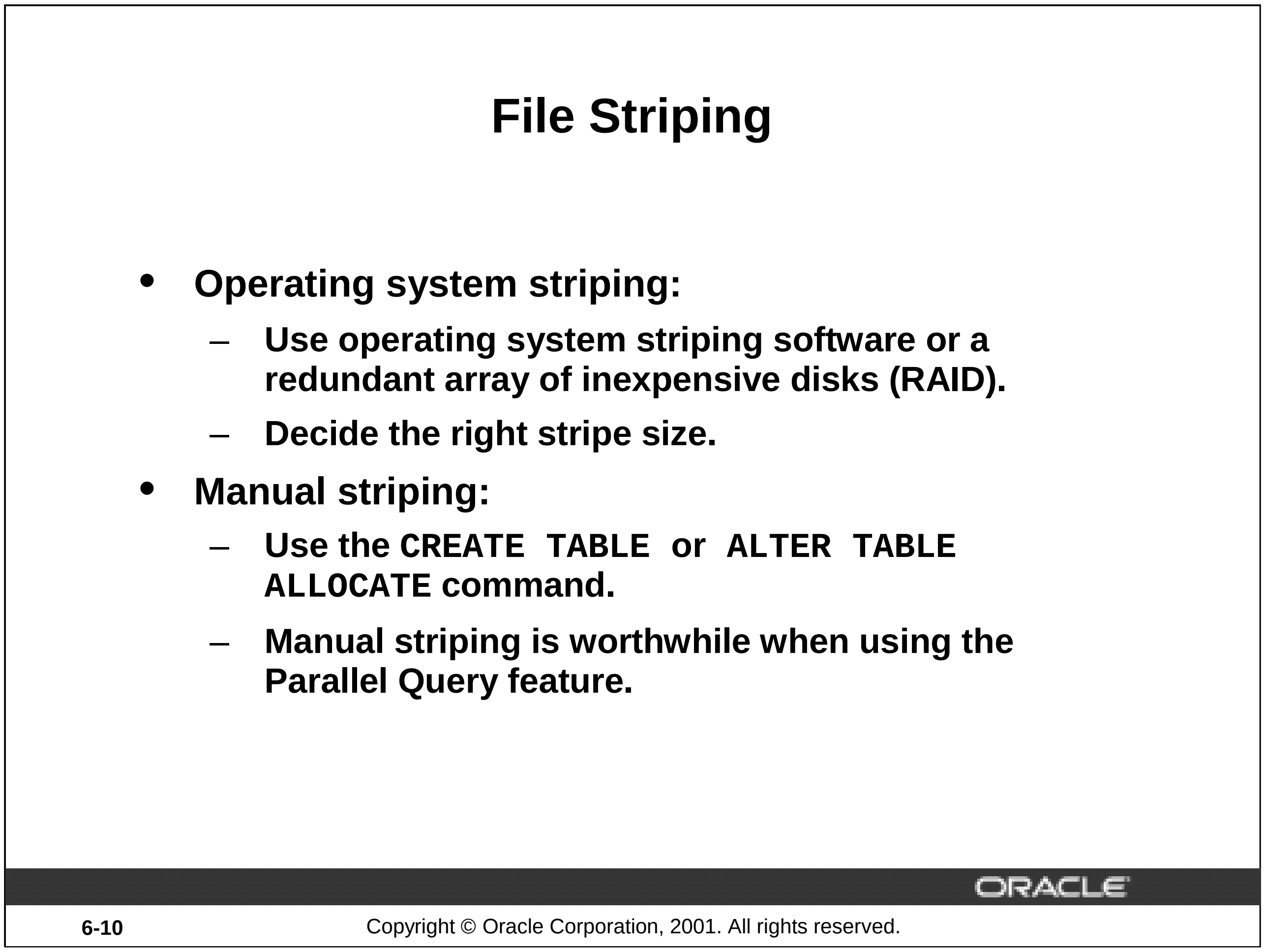


I/O Statistics

```
SQL> Rem I/O should be spread evenly across drives. A big difference between
phys_reads and phys_blks_rd implies table scans are going on.
SQL> select d.tablespace_name TABLESPACE, d.file_name, f.phyrds, f.phyblkrd,
  2> f.readtim, f.phywrts, f.phyblkwrt, f.writetim
  3> from v$filestat f, dba_data_files d
  4> where f.file# = d.file_id
  5> order by tablespace_name, file_name;
TABLESPACE    FILE_NAME              PHYRDS  PHYBLKRD  READTIM PHYWRTS PHYBLKWRT WRITETIM
------------- ---------------------- ------ --------- -------- ------- --------- --------
UNDO1         /u02/undots01.dbf          26        26       50     257       257      411
USERS         /u03/users01.dbf        65012    416752    38420     564       564     8860
SAMPLE        /u02/sample01.dbf           8         8        0       8         8        0
SYSTEM        /u01/system01.dbf         806      1538     1985     116       116     1721
TEMP          /u04/temp01.dbf           168       666      483     675       675        0
QUERY_DATA    /u01/query_data01.dbf       8         8        0       8         8        0
6 rows selected.
```

I/O Statistics

Check the output in STATSPACK to observe how well the I/O load is distributed across the disk devices. The output shows which files are most active. In the example shown on the slide, the USERS tablespace is being hit heavily. About 98% of the reads are being performed on the data file that contains tables. The index data file is having 0.01% of the disk reads performed against it.



File Striping

- **Operating system striping:**
  - **Use operating system striping software or a redundant array of inexpensive disks (RAID).**
  - **Decide the right stripe size.**
- **Manual striping:**
  - **Use the `CREATE TABLE` or `ALTER TABLE ALLOCATE` command.**
  - **Manual striping is worthwhile when using the Parallel Query feature.**

Operating System Striping

Your operating system may allow striping, in which what appears to be a single contiguous file is actually spread among devices. For example, you can use operating system striping software, such as a logical volume manager.

Choose the right stripe size, which depends on the Oracle block size and the `DB_FILE_MULTIBLOCK_READ_COUNT` initialization parameter.

The most common variety of striped file system in UNIX is the redundant array of inexpensive disks (RAID). Different levels of RAID striping have varying degrees of safety checks built in.

Manual Striping

You can create tablespaces so that they are made up of multiple files, each on a separate disk. You then create tables and indexes so that they are spread across these multiple files.

You can stripe by:

- Creating objects with `MINEXTENTS` greater than 1, where each extent is slightly smaller than the striped data files
- Allocating extents to a file explicitly:

```
ALTER TABLE tablename
ALLOCATE EXTENT (DATAFILE ’filename’ SIZE 10 M);
```

Operating System Striping (continued)

Using Striping

If your operating system offers striping, you should take advantage of it. You need to think about the size of the stripe, which should be a multiple of the value that you have set for `DB_FILE_MULTIBLOCK_READ_COUNT` (this will be discussed in a subsequent section).

With OS striped database files with a stripe width close to (or less than) `DB_FILE_MULTIBLOCK_READ_COUNT` x `DB_BLOCK_SIZE,` you may get two or more physical reads for each Oracle server read, because you have to access two or more disks.

Keep in mind that striping by hand is a labor-intensive task. The Oracle server fills the extents that you have created one after another. At any given time, one extent is likely to be *hot* and the others less active. If you are using the Parallel Query feature and doing many full table scans, then striping by hand may be worthwhile.

As with many other tuning issues, you can make the right choice only when you have thorough knowledge of how the data is used.

Controller-based striping will usually outperform operating system level striping. Multiple striped file systems can be employed instead of a single large striped set.

Tuning Full Table Scan Operations

- **Investigate the need for full table scans.**
- **Specify the `DB_FILE_MULTIBLOCK_READ_COUNT` initialization parameter in order to:**
  - **Determine the number of database blocks the server reads at once**
  - **Influence the execution plan of the cost-based optimizer**
- **Monitor long-running full table scans with `V$SESSION_LONGOPS` view.**

Investigating Full Table Scans

If there is high activity on one disk, it is often an untuned query causing the damage. The goal of performance tests is to increase the effectiveness with which data is accessed.

The following query provides an overview of how many full table scans are taking place:

```
SQL>  SELECT name, value FROM v$sysstat
 2    WHERE name LIKE ’%table scan%’;
NAME                                                 VALUE
------------------------------------------------     -----
table scans (short tables)                             125
table scans (long tables)                               30
table scans (rowid ranges                                0
table scans (cache partitions)                           0
table scans (direct read)                                0
table scan rows gotten                               21224
table scan blocks gotten                               804
7 rows selected.
```

Investigating Full Table Scans (continued)

The values for `table scans (long tables)` and `table scans (short tables)` relate to the full table scans.

If the value of `table scans (long tables)` is high, then a large percentage of the tables accessed were not indexed lookups. Your application may need tuning, or you should add indexes. Make sure that the appropriate indexes are in place, and valid.

Tuning Full Table Scans

The `DB_FILE_MULTIBLOCK_READ_COUNT` initialization parameter determines the maximum number of database blocks read in one I/O operation during a full table scan. The setting of this parameter can reduce the number of I/O calls required for a full table scan, thus improving performance.

I/O is a function of the operating system, so there are limits specific to the operating system imposed on the setting of this parameter. The server's ability to read multiple blocks is limited by the operating system upper limit on the number of bytes that can be read in a single I/O call.

On most platforms before 7.3, the maximum "read" memory chunk is 64 KB, so setting the `DB_FILE_MULTIBLOCK_READ_COUNT` parameter to 64 KB per `DB_BLOCK_SIZE` gives no extra performance benefit.

Most platforms 7.3 or later the limit of the `DB_FILE_MULTIBLOCK_READ_COUNT` parameter is operating system specific. In addition, this parameter is dynamic, so individual sessions can use `ALTER SESSION SET` command to set a larger size for batch-type work.

Setting the `DB_FILE_MULTIBLOCK_READ_COUNT` parameter dictates how many I/O calls are required to complete a table scan. For example, if `DB_FILE_MULTIBLOCK_READ_COUNT` is set to 16, and the Oracle block size is 4 KB, then a sequential scan of a 64 KB table can be read in one pass. This improves the speed of the table scan and overall query performance. The goal of setting the `DB_FILE_MULTIBLOCK_READ_COUNT` parameter is so that table scans are performed in fewer, larger I/Os. This is done by evaluating the number of blocks required to complete each table scan over time, then adjusting the parameter so that most scans can be performed in one I/O.

Tuning Full Table Scans (continued)

The total number of I/Os actually required to perform a full table scan also depends on other factors, such as the size of the table and whether Parallel Query is being used.

The cost-based optimizer uses all of these factors, including the `DB_FILE_MULTIBLOCK_READ_COUNT` parameter, to determine the cost of full table scans. The cost-based optimizer favors full table scans when the cost is lower than index scans.

Monitoring Full Table Scan Operations

Users and DBAs can monitor the progress of full table scans and get an idea of the estimated completion time. The Oracle server maintains statistics tracking the progress of such operations and makes it available to the users through the `V$SESSION_LONGOPS` dynamic performance view.

Note: The `DBMS_APPLICATION_INFO` package contains a procedure, `SET_SESSION_LONGOPS,` to populate the view from an application.

```
SQL> SELECT sid, serial#, opname,
  2  TO_CHAR(start_time,‘HH24:MI:SS’)AS START,
  3  (sofar/totalwork)*100 AS PERCENT_COMPLETE
  4  FROM v$session_longops;
SID SERIAL#    OPNAME      START       PERCENT_COMPLETE
--- ---------  ----------- ----------  ----------------
 8  219        TABLE SCAN  13:00:09    48.98098
```

Checkpoints

- **Checkpoints cause:**
  - **`DBW`*n* to perform I/O**
  - **`CKPT` to update the data file header and control file**
- **Frequent checkpoints:**
  - **Reduce instance recovery time**
  - **Decrease run-time performance**

Checkpoints

The `LGWR` background process writes to redo log groups in a circular fashion.

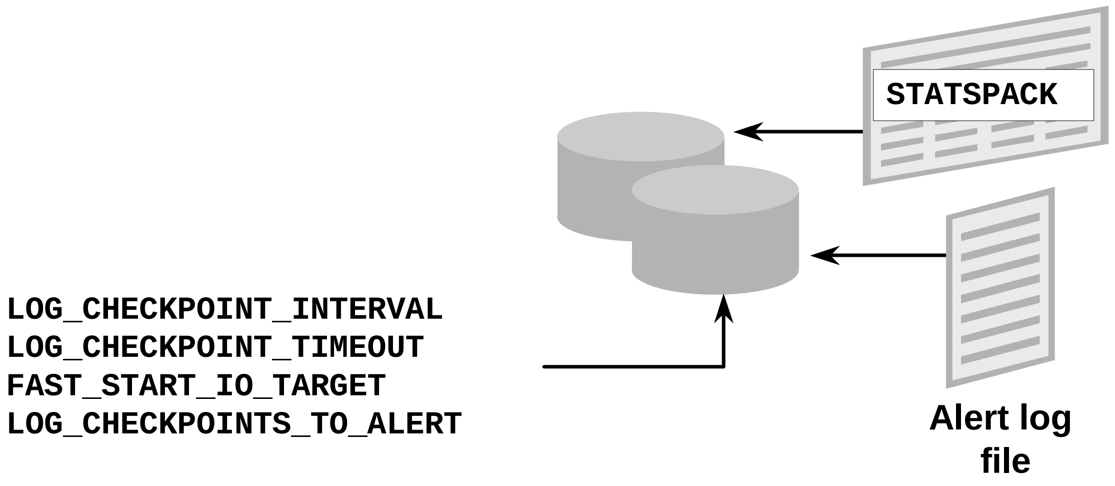
When a group is filled, the Oracle server must perform a checkpoint. This means that:

- `DBW`*n* writes all or part of the dirty blocks covered by the log being checkpointed to the data files.
- `CKPT` updates the data file headers and the control files.

Checkpoints normally allow other work to continue at the same time, although the checkpoint generates many disk writes. If the `DBW`*n* process has not finished checkpointing a file and `LGWR` needs the file again, `LGWR` has to wait.

Frequent checkpoints mean shorter instance recovery times but more writes by `DBW`*n* (to the data files) and `CKPT` (to the data file headers). Your choice depends on your priorities regarding recovery time and run-time performance.

Diagnostic Tools for Tuning Checkpoint Performance



Guidelines

- **Size the online redo log files to cut down the number of checkpoints.**
- **Add online redo log groups to increase the time before `LGWR` starts to overwrite.**
- **Regulate checkpoints with the following initialization parameters:**
  - **`FAST_START_IO_TARGET`**
  - **`LOG_CHECKPOINT_INTERVAL`**
  - **`LOG_CHECKPOINT_TIMEOUT`**
  - **`DB_BLOCK_MAX_DIRTY_TARGET`**
  - **`FAST_START_MTTR_TARGET`**

ORACLE

Monitoring Checkpoint Frequency

You can check the time between log switches in the `alert.log` file.

Also, check the file for error messages saying “Checkpoint not complete; unable to allocate file.” These messages mean that `LGWR` has waited for the checkpoint to finish.

If the system statistics Background Checkpoints Started and Background Checkpoints Completed have values that differ by more than 1, the checkpoints are not completing between log switches. You need larger log files.

You can set the `LOG_CHECKPOINTS_TO_ALERT` parameter so that the beginning and end of checkpoints are recorded in the `alert.log` file.

The statistic DBW*n* Checkpoint Write Requests indicates the number of times that checkpoints were send to the `DBW`*`n`*`.` A high number of checkpoints per transaction indicates that too many checkpoints are occurring.

Regulating Checkpoints

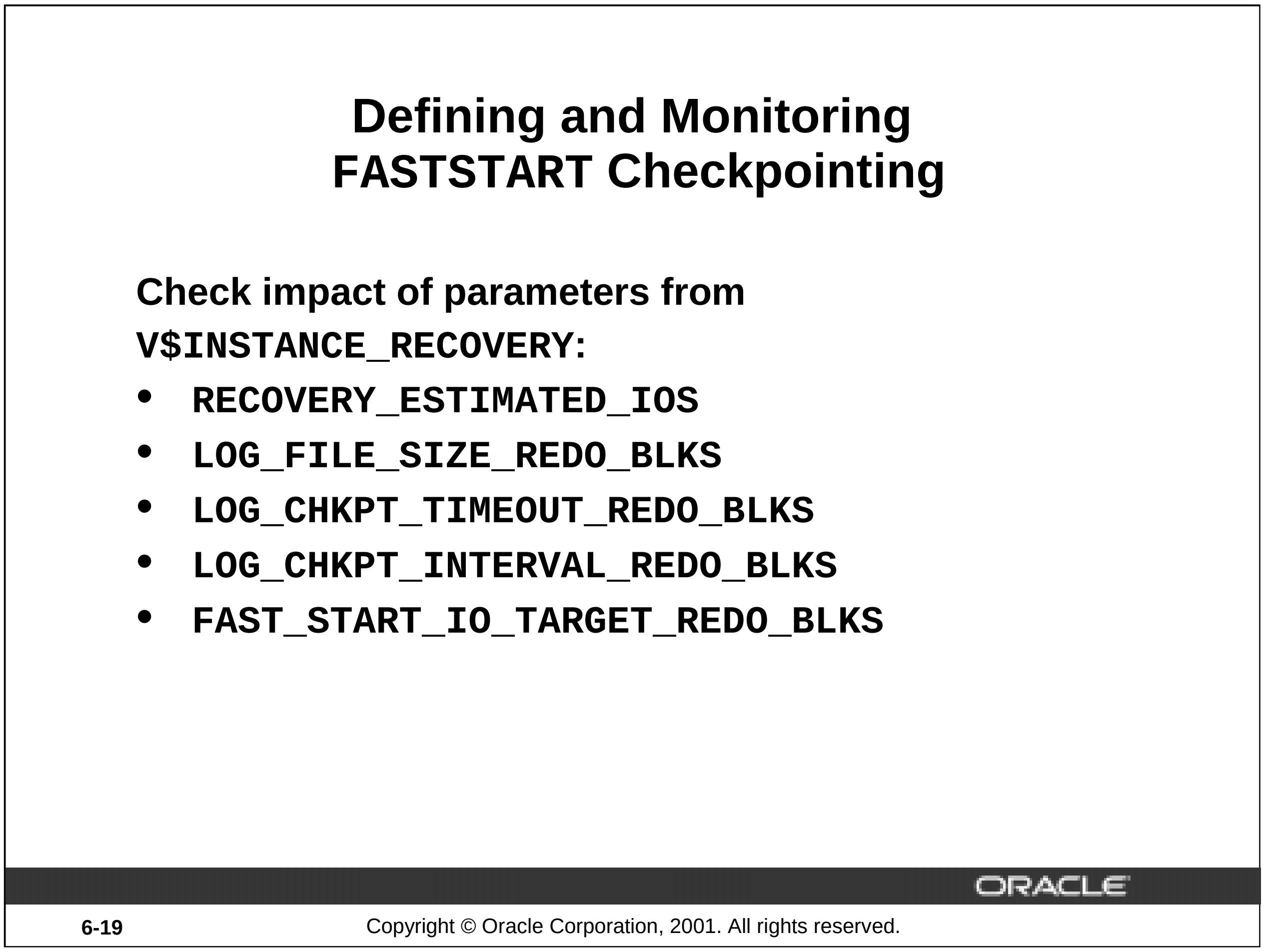
`DBW`*`n`* must always perform a checkpoint at the end of each redo log group. You can also control checkpoints with initialization parameters.

If efficient performance is your priority, choose a redo log file size such that checkpoints happen frequently enough not to cause a noticeable slowdown in response, but no more often than necessary.

Regulating Checkpoints (continued)

For many sites, this frequency is roughly every 30 minutes, but your database may have checkpoint frequencies of anything between two seconds and eight hours, depending on the database transaction volume.

You can experiment with different checkpoint frequencies. For instance, OLTP systems may experience disk contention during checkpoints if the SGA is very large and checkpoints are infrequent. In this case, more frequent checkpoints involve fewer dirty blocks.



Defining and Monitoring FASTSTART Checkpointing

Check impact of parameters from `V$INSTANCE_RECOVERY`:

- **`RECOVERY_ESTIMATED_IOS`**
- **`LOG_FILE_SIZE_REDO_BLKS`**
- **`LOG_CHKPT_TIMEOUT_REDO_BLKS`**
- **`LOG_CHKPT_INTERVAL_REDO_BLKS`**
- **`FAST_START_IO_TARGET_REDO_BLKS`**

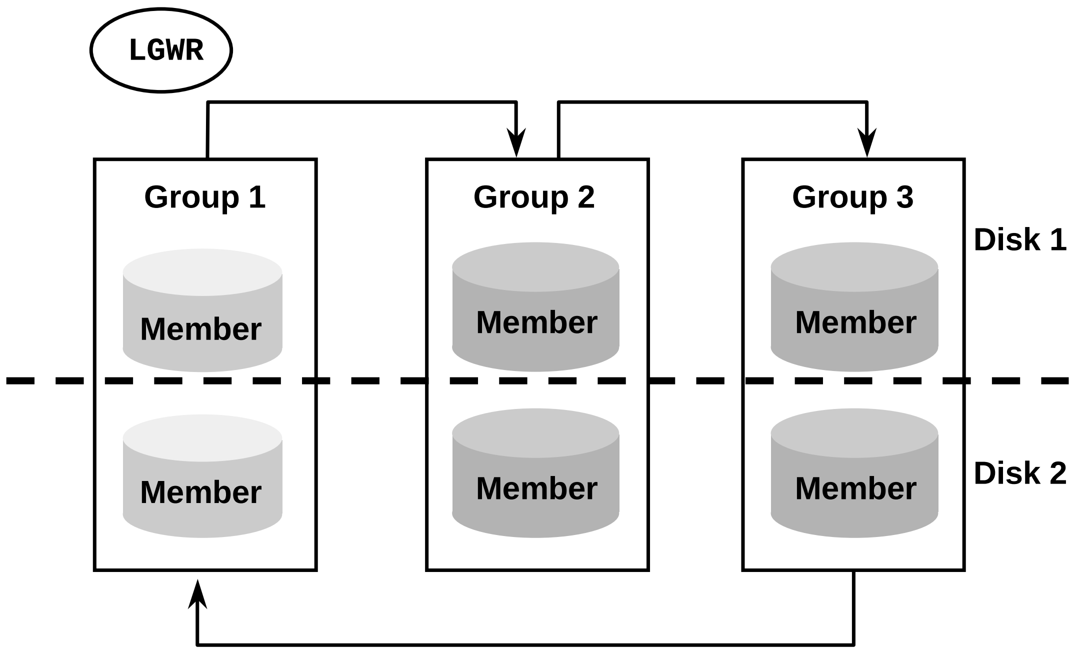
To specify a value for `FAST_START_IO_TARGET`, decide on the required service level after discussion with users. Translate this time to equivalent number of datafile I/O by using the average I/O time statistic from the view `V$FILESTAT`.

Monitoring Impact of Parameters on Recovery Time

Use the `V$INSTANCE_RECOVERY` view to obtain the following information:

- `RECOVERY_ESTIMATED_IOS`: The estimated number of data blocks to be processed during recovery based on the in-memory value of the fast-start checkpoint parameter
- `ACTUAL_REDO_BLKS`: The current number of redo blocks required for recovery
- `TARGET_REDO_BLKS`: The goal for the maximum number of redo blocks to be processed during recovery. This value is the minimum of the following 4 columns.
- `LOG_FILE_SIZE_REDO_BLKS:` The number of redo blocks to be processed during recovery to guarantee that a log switch never has to wait for a checkpoint
- `LOG_CHKPT_TIMEOUT_REDO_BLKS`: The number of redo blocks that need to be processed during recovery to satisfy `LOG_CHECKPOINT_TIMEOUT`
- `LOG_CHKPT_INTERVAL_REDO_BLKS` The number of redo blocks that need to be processed during recovery to satisfy `LOG_CHECKPOINT_INTERVAL`
- `FAST_START_IO_TARGET_REDO_BLKS`: The number of redo blocks that need to be processed during recovery to satisfy `FAST_START_IO_TARGET`

Redo Log Groups and Members



Redo Log Groups and Members

The above diagram shows one method of assigning redo log members to disk space. This method will support redo logging adequately. If archiving is enabled, it may be necessary to have groups on different disks as well as the members so that archiving will not contend with the redo log writer.

Online Redo Log File Configuration

- **Size redo log files to minimize contention.**
- **Provide enough groups to prevent waiting.**
- **Store redo log files on separate, fast devices.**
- **Query the `V$LOGFILE, V$LOG` and `V$LOG_HISTORY` dynamic performance views.**

6-21

Online Redo Log File Configuration

Online redo log files are organized in groups. A group must have one or more members. All members of a group have identical contents. You should have two or more members in each group for safety, unless you are mirroring all files at a hardware level. Redo log files in the same group should ideally be on separate, fast devices, because `LGWR` writes to them almost continuously. Properly size redo log files to minimize contention and frequent log switches. You can use the Redo Size statistics in the `STATSPACK` report.

Monitoring Redo Log File Information

You can query the `V$LOGFILE` and `V$LOG` dynamic performance views to obtain information about the name, location, size, and status of the online redo log file.

Any waits for Log File Parallel Write in `V$SYSTEM_EVENT` indicate a possible I/O problem with the log files.

Note: Redo log files can be created on raw devices.

The Oracle server does not provide a means of monitoring redo disk I/Os, so you must use operating system disk monitoring commands. On most UNIX systems, `sar` (system activity reporter) is useful for this purpose.

Monitoring Redo Log File Information (continued)

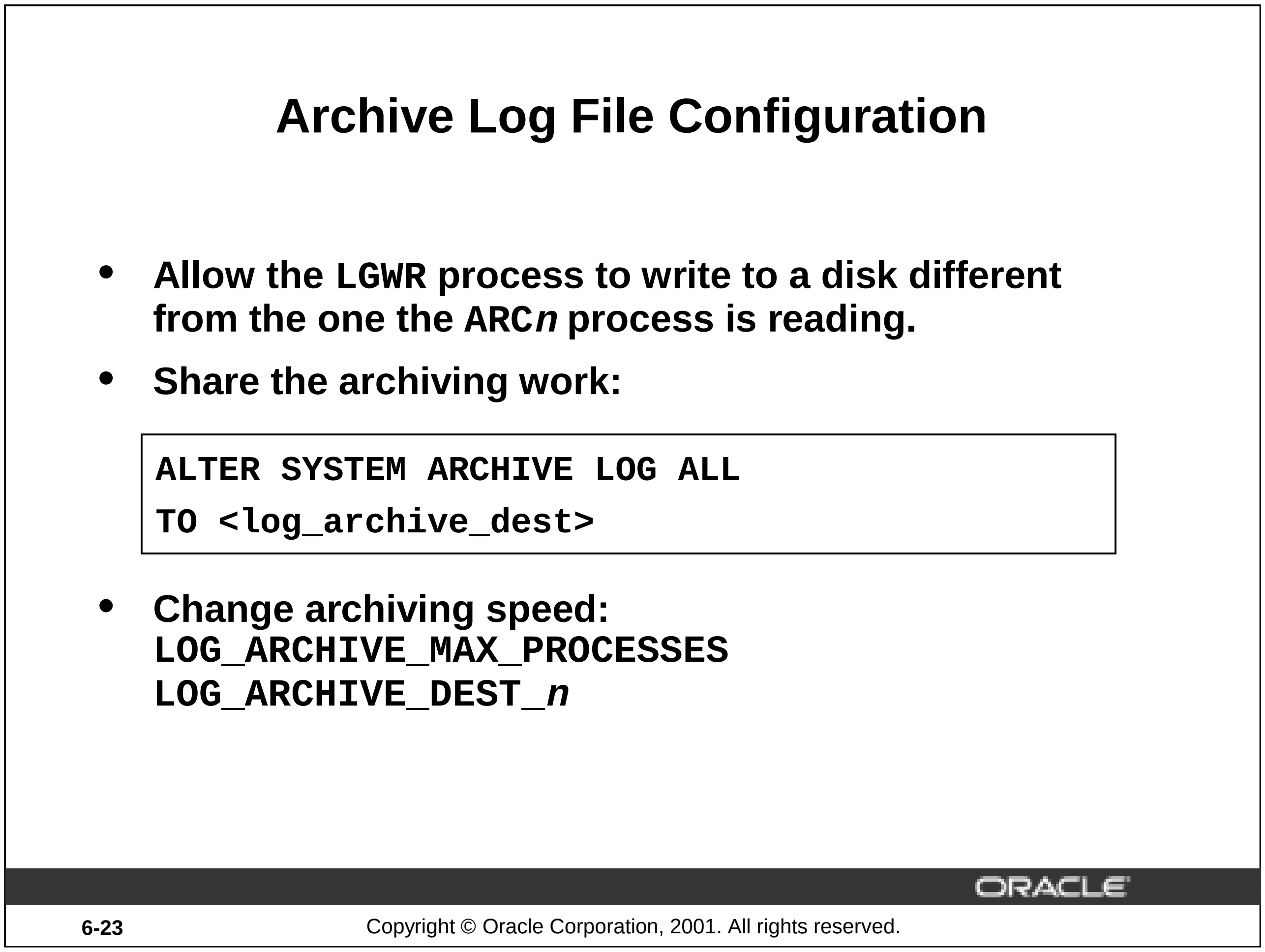
```
# sar -d 1 1

SunOS stc-sun101 5.6 Generic_105181-16 sun4u    02/01/01

22:30:03   device        %busy   avque   r+w/s  blks/s  avwait  avserv

22:30:04
           sd0              93     0.9     179     657     0.0     5.2
           sd0,a            93     0.9     179     657     0.0     5.2
           sd0,b             0     0.0       0       0     0.0     0.0
           sd0,c             0     0.0       0       0     0.0     0.0
           sd1               0     0.0       0       0     0.0     0.0
           sd1,c             0     0.0       0       0     0.0     0.0
           sd1,h             0     0.0       0       0     0.0     0.0
           sd6               0     0.0       0       0     0.0     0.0
```

`%busy` is the percentage of time the device was busy during the polling period, `avque` is the *average* queue size, `r+w/s` and `blks/s` are read and writes per second and blocks per second, respectively, `avwait` is the *average* wait time per request, and `avserv` is number of miliseconds per *average* seek. The `sar` command can be set up to record historical system information, making it even more useful to the DBA.

Another useful UNIX utility for monitoring disk activity is `iostat`. The output of `iostat` is simpler than that of `sar`:

```
# iostat -D
          sd0           sd1           sd6          nfs1
 rps wps util  rps wps util  rps wps util  rps wps util
   2   1  1.4    7   2  6.0    0   0  0.0    0   0  0.0
```

`rps` and `wps` are reads per second and writes per second, respectively, and `util` is the percentage of disk utilization.



Archive Log File Configuration

- **Allow the `LGWR` process to write to a disk different from the one the `ARC`*`n`* process is reading.**
- **Share the archiving work:**

```
ALTER SYSTEM ARCHIVE LOG ALL
TO <log_archive_dest>
```

- **Change archiving speed:**
  `LOG_ARCHIVE_MAX_PROCESSES`
  `LOG_ARCHIVE_DEST_`*`n`*

Archive Log File Configuration

If you choose to archive, it is even more important to have more than two redo log groups.When a group switches to another group, the `DBW`*`n`* process must checkpoint as usual, and one file must be archived. You need to allow time for both of these operations before the `LGWR` process needs to overwrite the file again.

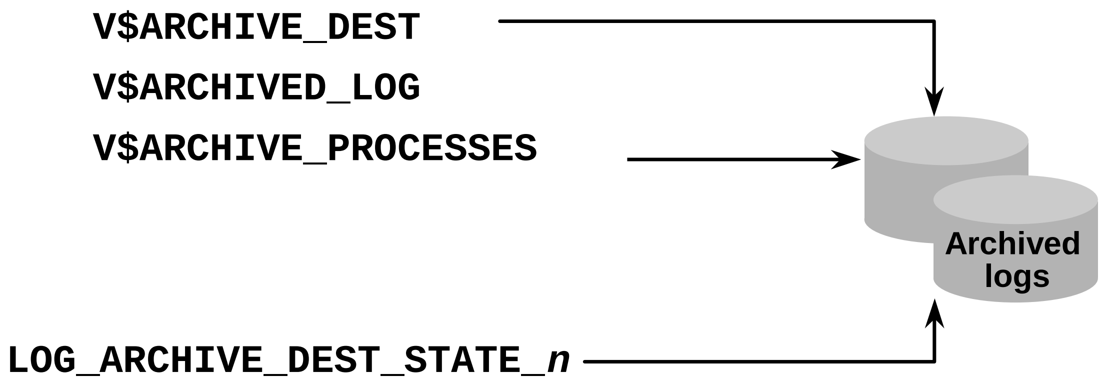
Obtaining Information about Archived Log Files and Their Location

You can query the `V$ARCHIVED_LOG` dynamic performance view to display archived log information from the control file, including archive log names. An archive log record is inserted after the online redo log is successfully archived or cleared (the name column is null if the log was cleared). If the log is archived twice, there will be two archived log records with the same thread number, sequence number, and first change number, but with different names.

The `V$ARCHIVE_DEST` dynamic performance view describes, for the current instance, all the archive log destinations, as well as their current values, modes, and statuses.

Note: The `LOG_ARCHIVE_DEST_`*`n`* parameter is valid only if you have installed the Oracle Enterprise Edition. You can continue to use `LOG_ARCHIVE_DEST` if you have installed the Oracle Enterprise Edition. However, you cannot use both `LOG_ARCHIVE_DEST_`*`n`* and `LOG_ARCHIVE_DEST` because they are not compatible.

Diagnostic Tools



Regulating Archiving Speed

- Occasionally, in busy databases, a single `ARC0` process cannot keep up with the volume of information written to the redo logs. Oracle9*i* allows the DBA to define multiple archive processes by using the `LOG_ARCHIVE_MAX_PROCESSES` parameter.
- The `LGWR` process starts a new `ARCn` process whenever the current number of `ARCn` processes is insufficient to handle the workload. If you anticipate a heavy workload for archiving, you can get another process to share the work by regularly running a script containing the command:

```
SQL> ALTER SYSTEM ARCHIVE LOG ALL TO 'directory_name';
```

- Monitor `V$ARCHIVE_PROCESSES` and spawn extra archiver processes whenever there are more than two logs that need archiving.

```
SQL> select * from v$archive_processes;
PROCESS          STATUS    LOG SEQUENCE        STAT
-------          --------  -------------       --------
    0 ACTIVE     122       BUSY
    1 ACTIVE     0         IDLE
    2 STOPPED    0         IDLE
```

Note: The number of processes used by the `ARC` process to archive redo log files is automatically set to 4 when the `DBWR_IO_SLAVES` initialization parameter is set to a value greater than 0. Archive redo log files cannot be created on raw devices.

Summary

In this lesson, you should have learned how to:

- **List the advantages of partitioning different Oracle file types**
- **List reasons for partitioning data in tablespaces**
- **Diagnose tablespace usage problems**
- **Describe how checkpoints work**
- **Monitor and tune checkpoints**
- **Monitor and tune archive logging**

6-25

Practice 6

1. Connect as system/manager and diagnose database file configuration by querying the V$DATAFILE, V$LOGFILE and V$CONTROLFILE dynamic performance views.
2. Diagnose database file usage by querying the V$FILESTAT dynamic performance view, combine with V$DATAFILE in order to get the datafile names
3. Determine if there are waits for redo log files by querying the V$SYSTEM_EVENT dynamic performance view, where the waiting event is 'log file sync' or 'log file parallel write'

   Waits for:
   log file sync are indicative of slow disks that store the online logs, or un-batched commits.
   log file parallel write is much less useful. The reason it is less useful is that this event only shows how often LGWR waits, not how often server processes wait. If LGWR waits without impacting user processes, there is no performance problem. If LGWR waits, it is likely that the log file sync event (mentioned above) will also be evident.
4. Connect as perfstat/perfstat and diagnose file usage from STATSPACK
   - Generate a Statspack report using $HOME/STUDENT/LABS/spreport.sql.
   - Locate and open the report file.
   - Examine the report, and search for the string "File IO Stats"

   Note: On a production database care would be taken in monitoring the disk, and controller usage by balancing the workload across all devices. If your examination shows a distinct over utilization of a particular datafile, consider resolving the cause of the amount of I/O, for example investigate the number of full table scans, clustering of files on a specific device, and under utilization of indexes. If after this the problem remains then look at placing the datafile on a low utilization device.
5. Connect as system/manager and enable checkpoints to be logged in the alert file by setting the value of the parameter log_checkpoints_to_alert to true using "alter system set" command.
6. Connect as sh/sh and execute the $HOME/STUDENT/LABS/lab06_06.sql script to provide a workload against the database.
7. At the operation system level use the editor to open the alert log file (located in the directory specified by BACKGROUND_DUMP_DEST). Then determine the checkpoint frequency for your instance by searching for messages containing the phrase "Completed Checkpoint". The time difference between two consecutive messages is the checkpoint interval
8. Open the alert log file using an editor, and search for the line: Completed checkpoint

   The line before this will be the time at which the checkpoint occurred. Search for the following checkpoint time, and then subtract to get the time between checkpoints.

   Note: On a production system the checkpoint interval should range between 15 minutes to 2 hours. The actual interval is dependant on the type of application, and the amount DML activity.

7

Optimizing Sort Operations

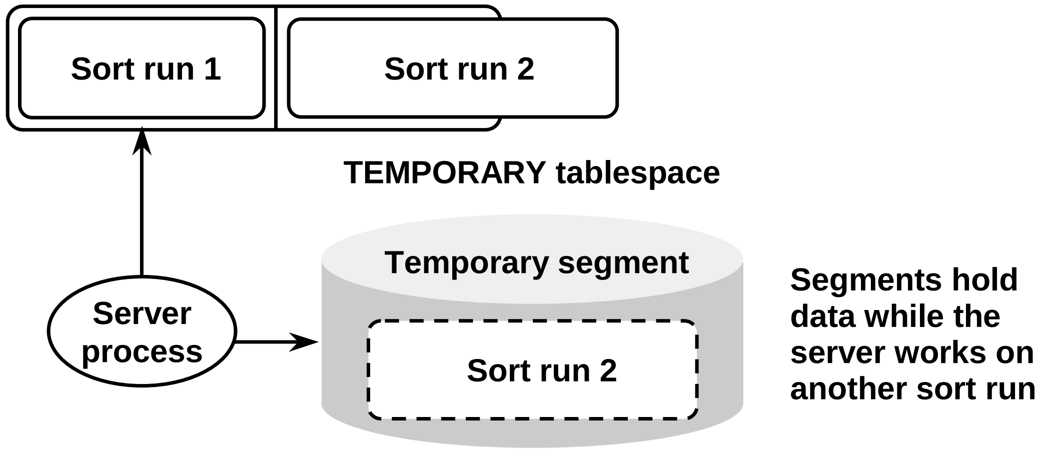
Objectives

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

- Describe how sorts are performed
- Identify the SQL operations that require sorts
- Differentiate between disk and memory sorts
- Create and monitor TEMPORARY Tablespaces
- List ways to reduce total sorts and disk sorts
- Determine the number of sorts performed in memory
- Set old and new sort parameters

The Sorting Process

If sort space requirement is greater than `SORT_AREA_SIZE`:



The Sorting Process

The server sorts in memory if the work can be done within an area smaller than the value (in bytes) of the `SORT_AREA_SIZE` parameter.

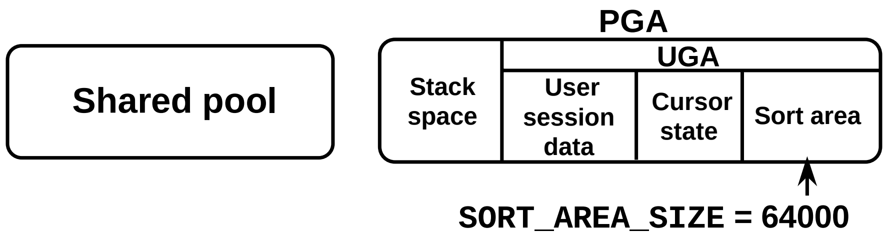
If the sort needs more space than this value:

1. The data is split into smaller pieces, called sort runs; each piece is sorted individually.
2. The server process writes pieces to temporary segments on disk; these segments hold intermediate sort run data while the server works on another sort run.
3. The sorted pieces are merged to produce the final result. If `SORT_AREA_SIZE` is not large enough to merge all the runs at once, subsets of the runs are merged in a number of merge passes.

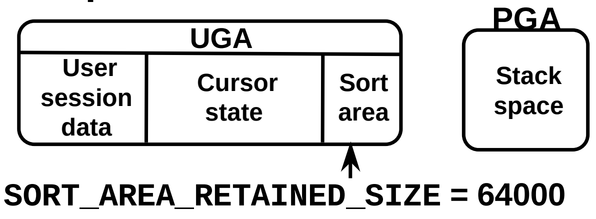
Sort Area and Parameters

The sort space is in:

- **The PGA for a dedicated server connection**



- **The shared pool for Oracle Share Server connection**



Sort Area

The sort space:

- Is part of the PGA when connected with a dedicated server
- Is part of the shared pool when connected with Oracle Shared Server

Note: An application doing large sorts should not be using Oracle Shared Server.

Parameters

`SORT_AREA_SIZE`

- The sort area is sized with the `SORT_AREA_SIZE` `init.ora` parameter.
- It can be set dynamically, using the `ALTER SESSION` or `ALTER SYSTEM DEFERRED` command.
- The default value is dependent on the operating system.
- The default is generally adequate for most OLTP operations. Adjust it upward for DSS applications, batch jobs, or large operations.

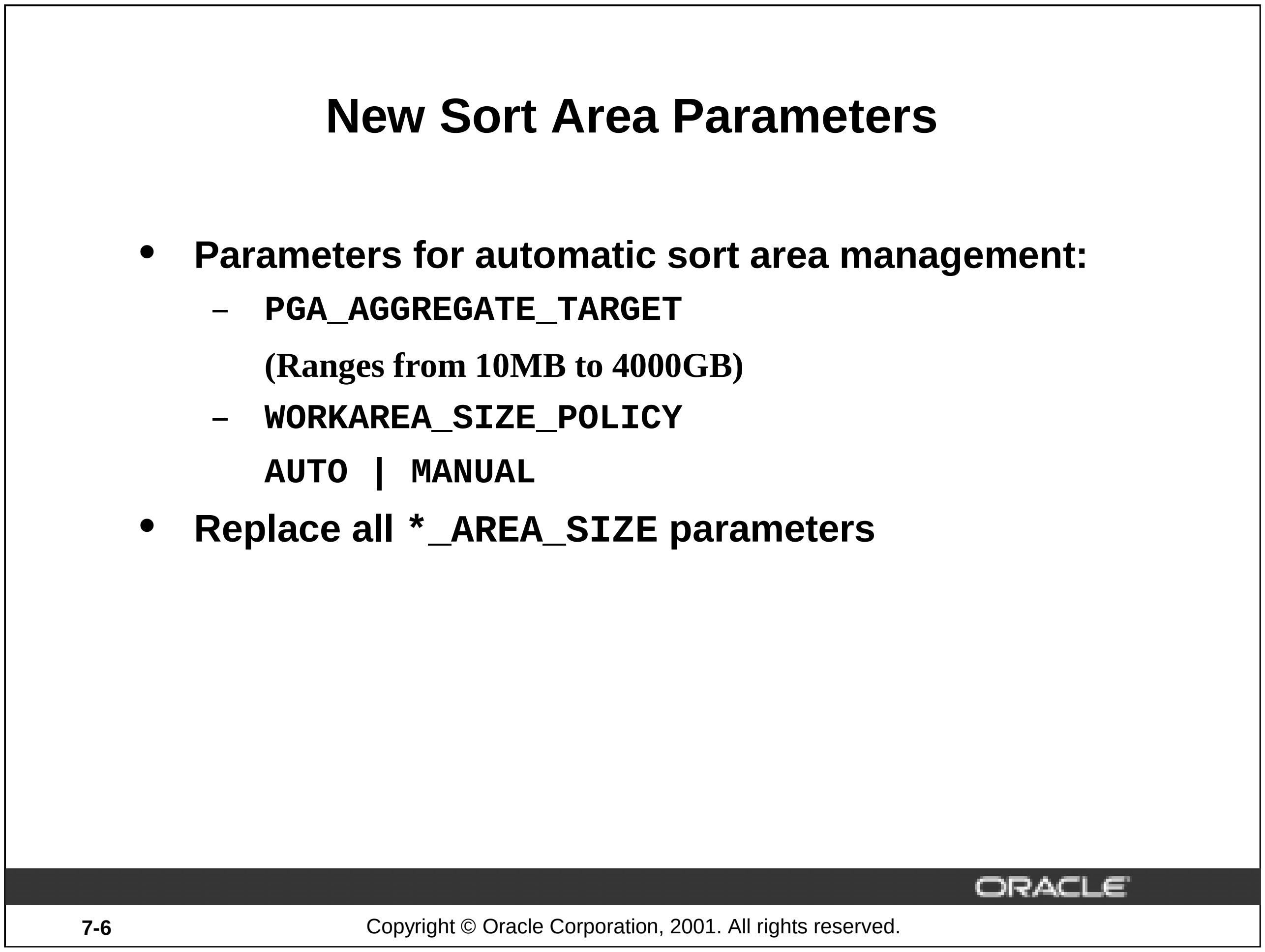
Parameters (continued)

`SORT_AREA_RETAINED_SIZE`

- When the sorting is complete and the sort area still contains sorted rows to be fetched, the sort area can shrink to the size specified by the `SORT_AREA_RETAINED_SIZE` parameter.
- The memory is released back to the UGA for use by the same Oracle server process (not to the operating system) after the last row is fetched from the sort space.
- The default value for this parameter is equal to the value of the `SORT_AREA_SIZE` parameter.

Initialization Parameters for Bitmap Indexing:

- `CREATE_BITMAP_AREA_SIZE`:
  - Value is static, thus can only be changed by restarting the database
  - Amount of memory allocated for bitmap creation
  - Not dynamically alterable at the session level
  - Default value: 8 MB (A larger value may lead to faster index creation. If the cardinality is very small, however, you can set a small value for this parameter. For example, if the cardinality is only two, then the value can be on the order of kilobytes. As a general rule, the higher the cardinality, the more memory is needed for optimal performance.)
- `BITMAP_MERGE_AREA_SIZE`:
  - Value is static, thus can only be changed by restarting the database
  - Amount of memory used to merge bitmaps retrieved from a range scan of the index
  - Not dynamically alterable at the session level
  - Default value: 1 MB (A larger value should improve performance, because the bitmap segments must be sorted before being merged into a single bitmap.)



New Sort Area Parameters

- **Parameters for automatic sort area management:**
  - **`PGA_AGGREGATE_TARGET`**

    (Ranges from 10MB to 4000GB)
  - **`WORKAREA_SIZE_POLICY`**

    `AUTO | MANUAL`
- **Replace all `*_AREA_SIZE` parameters**

7-6

New Sort Area Parameters

`PGA_AGGREGATE_TARGET`

`PGA_AGGREGATE_TARGET` specifies the target aggregate PGA memory of all server processes attached to the instance. The value of this parameter ranges from 10MB to 4000GB.

When setting this parameter, you should examine the total memory on your system that is available to the Oracle instance and subtract the SGA, then assign the remaining memory to `PGA_AGGREGATE_TARGET`.

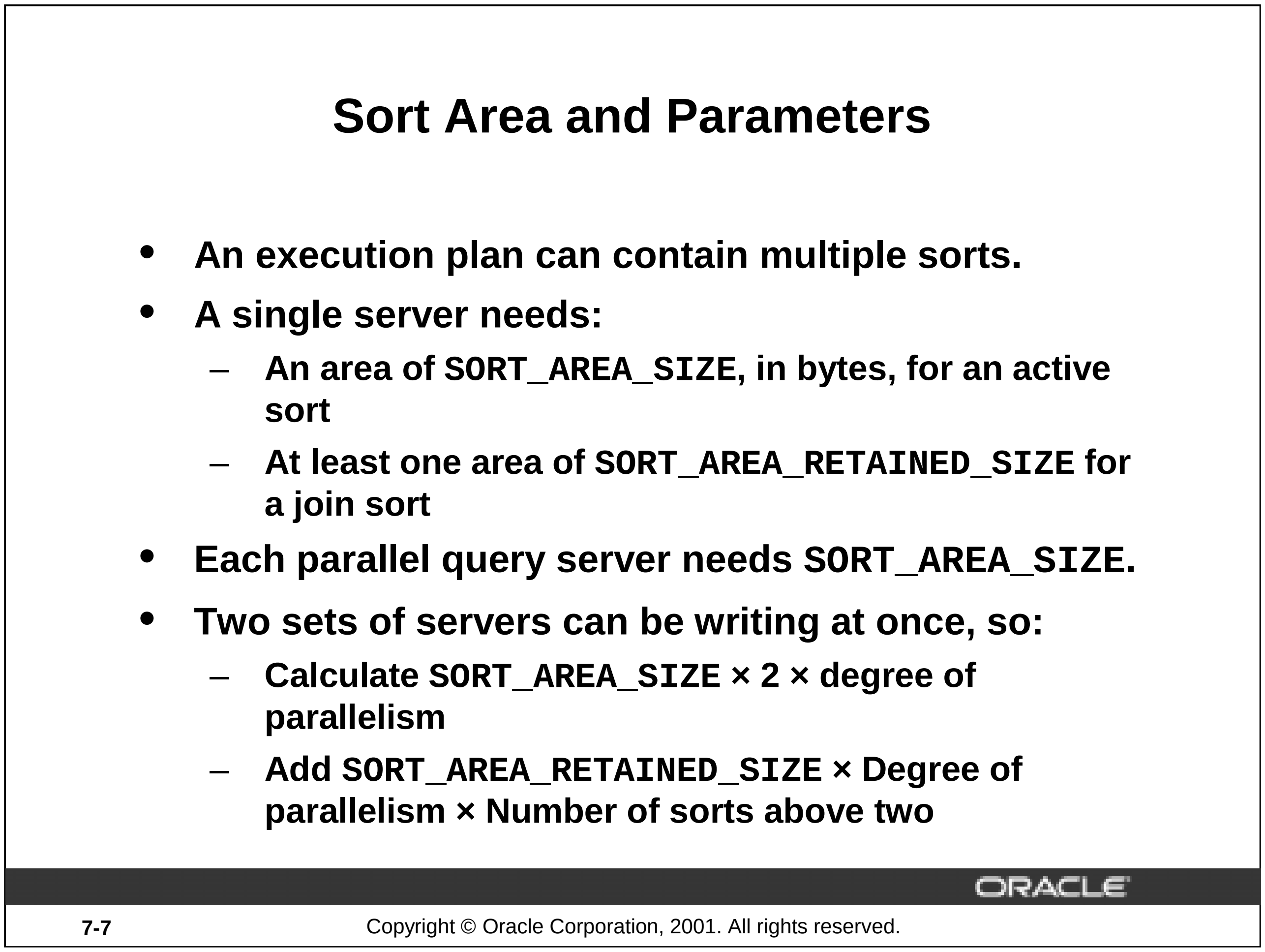
`WORKAREA_SIZE_POLICY`

Accepted values are:

- AUTO

  You can specify AUTO only when `PGA_AGGREGATE_TARGET` is defined.
- MANUAL

  The sizing of work areas is manual and based on the values of the `*_AREA_SIZE` parameter corresponding to the operation (for example, a sort uses `SORT_AREA_SIZE`). Specifying MANUAL may result in sub-optimal performance and poor PGA memory utilization.



Sort Area and Parameters

- An execution plan can contain multiple sorts.
- A single server needs:
  - An area of `SORT_AREA_SIZE`, in bytes, for an active sort
  - At least one area of `SORT_AREA_RETAINED_SIZE` for a join sort
- Each parallel query server needs `SORT_AREA_SIZE`.
- Two sets of servers can be writing at once, so:
  - Calculate `SORT_AREA_SIZE` × 2 × degree of parallelism
  - Add `SORT_AREA_RETAINED_SIZE` × Degree of parallelism × Number of sorts above two

7-7

Memory Requirements

Single Server Process

An execution plan can contain multiple sorts. For example, a sort-merge join of two tables may be followed by a sort for an `ORDER BY` clause. This gives three sorts all together.

If a single server works on the sort, then while it does the `ORDER BY` sort it uses:

- An area of `SORT_AREA_SIZE,` in bytes, for the active sort
- Two areas of the size specified by `SORT_AREA_RETAINED_SIZE,` for the join sorts

Parallel Query Processes

If you parallelize the statement, each query server needs `SORT_AREA_SIZE` amount of memory.

With parallel query, two sets of servers can be working at once, so you should:

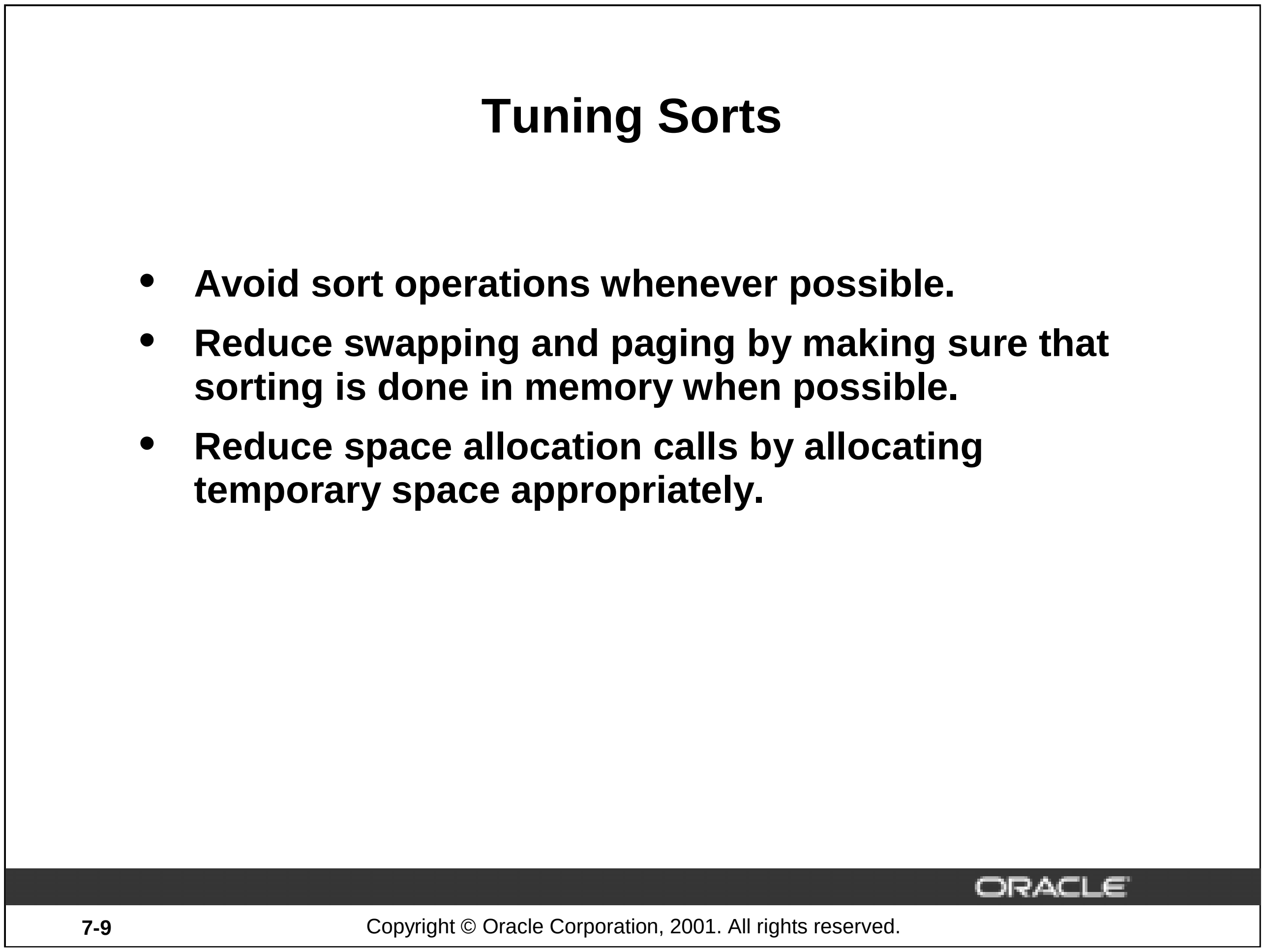
- Calculate `SORT_AREA_SIZE` × 2 × Degree of parallelism
- If necessary, add `SORT_AREA_RETAINED_SIZE` × Degree of parallelism × Number of sorts above two

If you can still afford the memory, the optimal value for `SORT_AREA_SIZE` and `SORT_AREA_RETAINED_SIZE` with Parallel Query is one megabyte. In testing, larger values than this have not improved performance significantly.

Sizes

Usually, `SORT_AREA_SIZE` and `SORT_AREA_RETAINED_SIZE` should be set to the same value, unless:

- You are very short of memory
- You are using Oracle Shared Server



Tuning Sorts

- **Avoid sort operations whenever possible.**
- **Reduce swapping and paging by making sure that sorting is done in memory when possible.**
- **Reduce space allocation calls by allocating temporary space appropriately.**

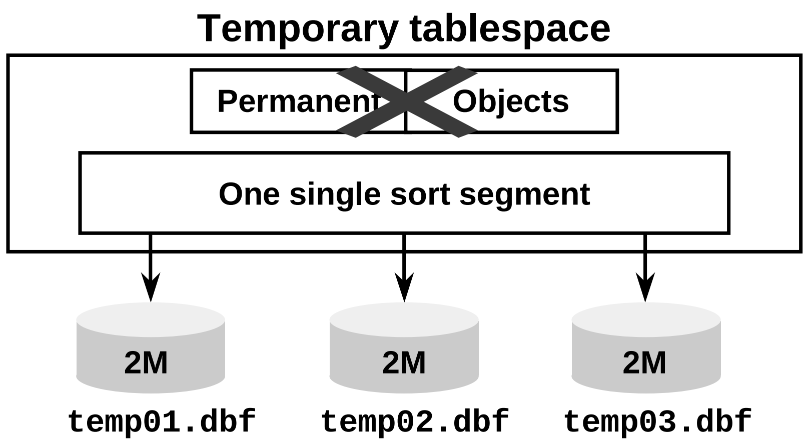
Diagnosing Problems

- You may be able to avoid sorts if the processed data is sorted previously.
- When sorts are not too large, a sort area that is too small results in performance overheads by swapping to disk; ensure that the sort operations occur in memory.
- Using large chunks of memory for sorting may result in paging and swapping, and reduce overall system performance.
- Allocation and deallocation of temporary segments occur frequently on permanent tablespaces.

Tuning Goals

- Avoiding sort operations that are not necessary
- Optimizing memory sort and disk overhead
- Eliminating space allocation calls to allocate and deallocate temporary segments

The Sorting Process and Temporary Space



Create a temporary tablespace using:

```
CREATE TEMPORARY TABLESPACE TEMP TEMPFILE
'$HOME/ORADATA/u06/temp01.dbf' size 200M;
```

Advantage of Temporary Tablespaces

Designating temporary tablespaces exclusively for sorts effectively eliminates serialization of space management operations involved in the allocation and deallocation of sort space.

A temporary tablespace:

- Cannot contain any permanent objects
- Contains a single sort segment per instance for Oracle Parallel Server environments

A temporary tablespace has to have temporary files. The advantage of these files is that they do not need to be a part of the backup strategy, thus saving time during the backup process.

Temporary Space Segments

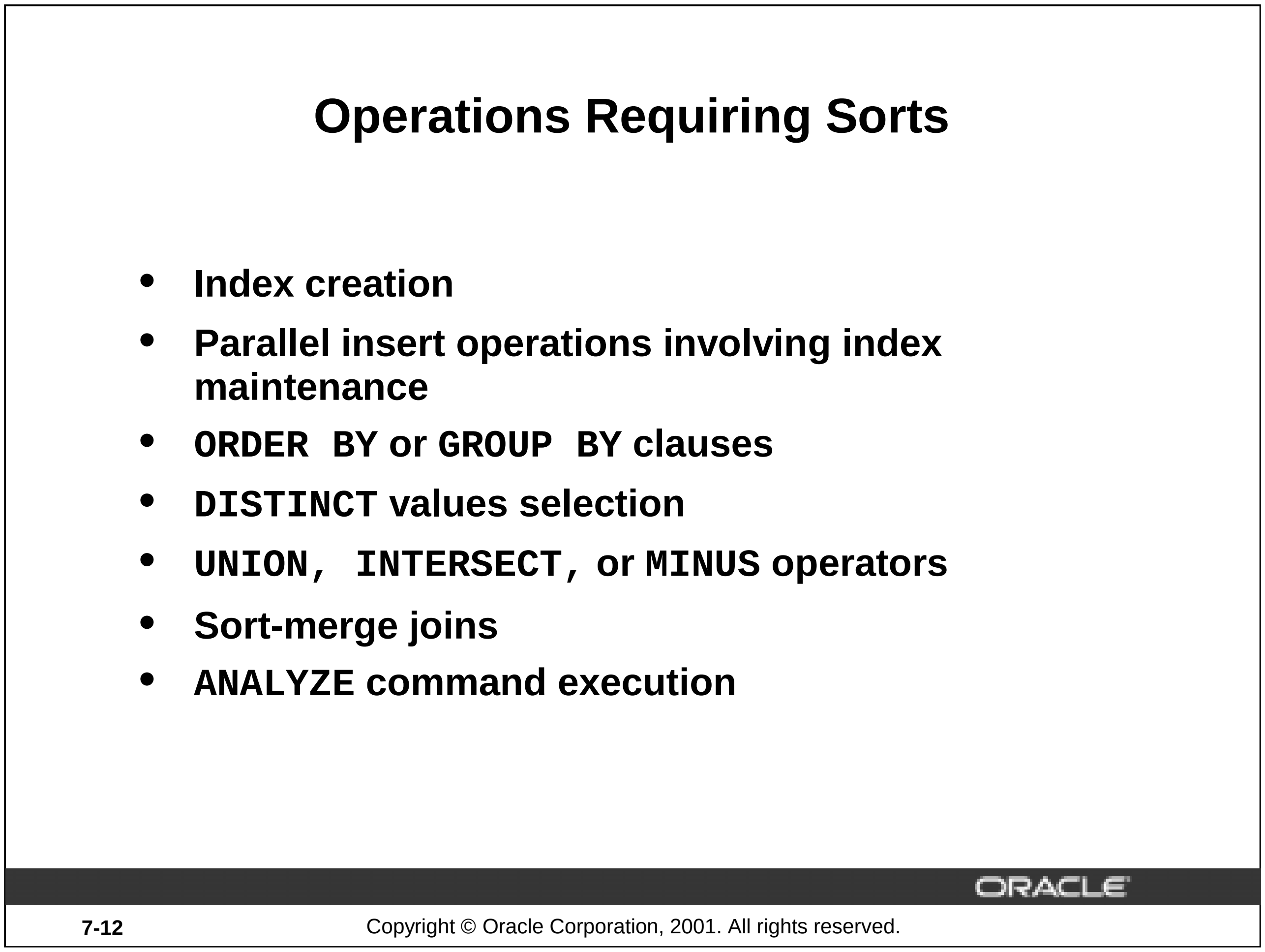
A temporary space segment:

- **Is created by the first sort**
- **Extends as demands are made on it**
- **Comprises extents, which can be used by different sorts**
- **Is described in SGA in the sort extent pool (SEP)**

ORACLE

The Sort Segment

- Created at the time of the first sort operation that uses the tablespace
- Dropped when the database is closed
- Grows as demands are made on it
- Made up of extents, each of which can be used by different sort operations
- Described in an SGA structure called the sort extent pool (SEP). When a process needs sort space, it looks for free extents in the SEP.



Operations Requiring Sorts

- Index creation
- Parallel insert operations involving index maintenance
- `ORDER BY` or `GROUP BY` clauses
- `DISTINCT` values selection
- `UNION, INTERSECT,` or `MINUS` operators
- Sort-merge joins
- `ANALYZE` command execution

7-12 Copyright © Oracle Corporation, 2001. All rights reserved.

Index Creation

The server process (or processes, if the index is being created in parallel) has to sort the indexed values before building the B-tree.

`ORDER BY` or `GROUP BY` Clauses

The server process must sort on the values in the `ORDER BY` or `GROUP BY` clauses.

`DISTINCT` Values

For the `DISTINCT` keyword, the sort has to eliminate duplicates.

`UNION`, `INTERSECT`, or `MINUS` Operators

Servers need to sort the tables they are working on to eliminate duplicates.

Sort-Merge Joins

```
SQL> select department_name, Last_name
 2>  from employees e, departments d
 3>  where e.department_id = d.department_id;
```

Sort-Merge Joins (continued)

If there are no indexes available, an equijoin request needs to:

- Perform full table scans of `EMPLOYEES` and `DEPARTMENTS` tables.
- Sort each row source separately.
- Merge the sorted sources together, combining each row from one source with each matching row of the other source.

Before the sort operation:

```
SQL> select name, value
    2   from v$sysstat where name = ’sorts (rows)’;
    NAME                                          VALUE
    ----------------- ------------------------------
    sorts (rows)                                  639330
```

After the sort operation:

```
SQL> select name, value
2   from v$sysstat where name = ’sorts (rows)’;
NAME                                               VALUE
----------------- ------------------------------
sorts (rows)                                    655714
```

`DBMS_STATS` Execution

The `DBMS_STATS` package is useful for collecting statistics on tables, indexes, and clusters to help the CBO define the best execution plans. It sorts the data to provide summarized information.

```
SQL> execute sys.dbms_utility.analyze_schema(‘HR','COMPUTE');
  PL/SQL procedure successfully completed.
```

Avoiding Sorts

Avoid sort operations whenever possible:

- **Use `NOSORT` to create indexes.**
- **Use `UNION ALL` instead of `UNION`.**
- **Use index access for table joins.**
- **Create indexes on columns referenced in the `ORDER BY` clause.**
- **Select the columns for analysis.**
- **Use `ESTIMATE` rather than `COMPUTE` for large objects.**

The `NOSORT` Clause

Use the `NOSORT` clause when creating indexes for presorted data on a single-CPU machine using SQL*Loader. This clause is only valid for the data inserted into a table:

```
SQL> create index EMPLOYEES_DEPARTMENT_ID_FK on
  employees(department_id) NOSORT;
ORA-01409: NOSORT option may not be used; rows are not in
  ascending order
```

On a multi-CPU machine, it is probably quicker to load data in parallel, even though it is not loaded in order. Then you can use parallel index creation to speed up sorting.

`UNION ALL`

Use `UNION ALL` instead of `UNION;` this clause does not eliminate duplicates, so does not need to sort.

Nested Loop Joins

Use index access for equijoin requests:

```
SQL> select department_name, Last_name
 2>  from employees e, departments d
 3>  where e.department_id = d.department_id;
```

Nested Loop Joins (continued)

The optimizer chooses a nested loop join instead of a sort-merge join. A nested loop join does not require any sorts. The steps necessary to do this are:

1. Perform a full table scan of the `employees` table.
2. Use the `DEPARTMENT_ID` value for each row returned to perform a unique scan on the `primary key` index.
3. Use the `ROWID` retrieved from the index scan to locate the matching row in the `departments` table.
4. Combine each row returned from `employees` with the matching row returned from `departments`.

Indexes and `ORDER BY`

Create indexes on columns that are frequently referenced with `ORDER BY` statements. Since the index is sorted in ascending order and is doubly linked, the server will use the index rather a sort operation.

`ANALYZE FOR COLUMNS`

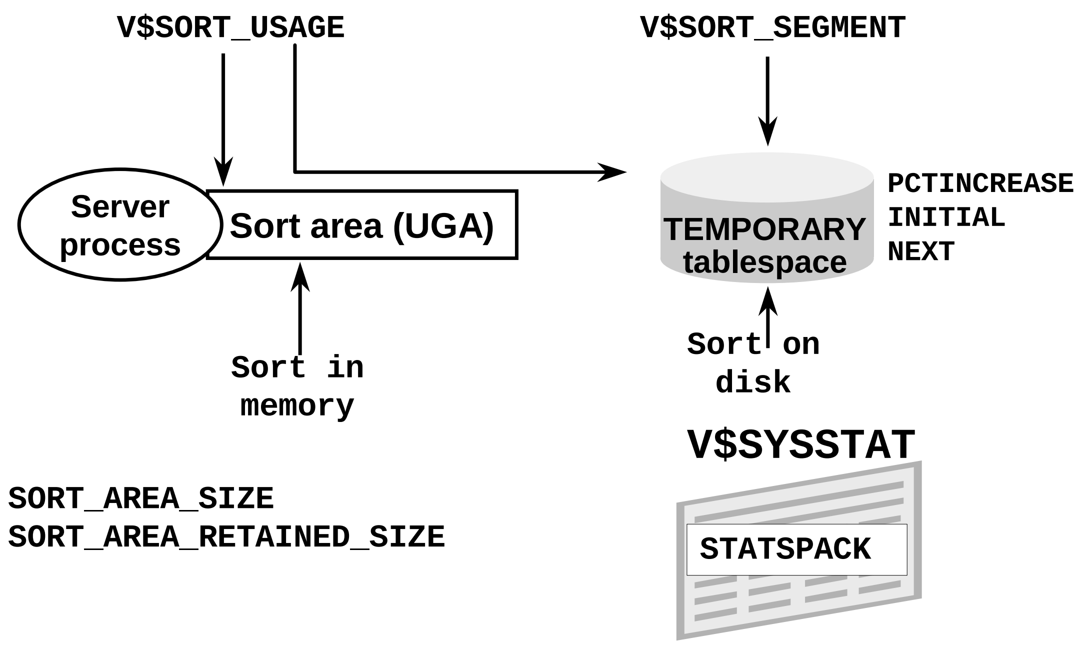
Collect statistics for the columns of interest only; for example, those involved in join conditions, `ANALYZE... FOR COLUMNS` or `ANALYZE...FOR ALL INDEXED COLUMNS.`

Note: The `ANALYZE... SIZE` *`n`* command creates histograms for the columns involved. Combining this clause with the `FOR ALL INDEXED COLUMNS` clause generates unnecessary histograms for primary keys and unique constraints.

`ANALYZE ESTIMATE`

The `COMPUTE` clause is more precise for the optimizer. However, it demands a large amount of sort space. The `ESTIMATE` clause is preferable for large tables and clusters.

Diagnostic Tools



Dynamic Views and STATSPACK Output

The V$SYSSTAT view displays the number of sorts in memory, sorts on disk, and rows being sorted:.

- `Sorts (disk):` Number of sorts requiring I/O to temporary segments
- `Sorts (memory):` Number of sorts performed entirely in memory
- `Sorts (rows):` Total rows sorted in the period being monitored

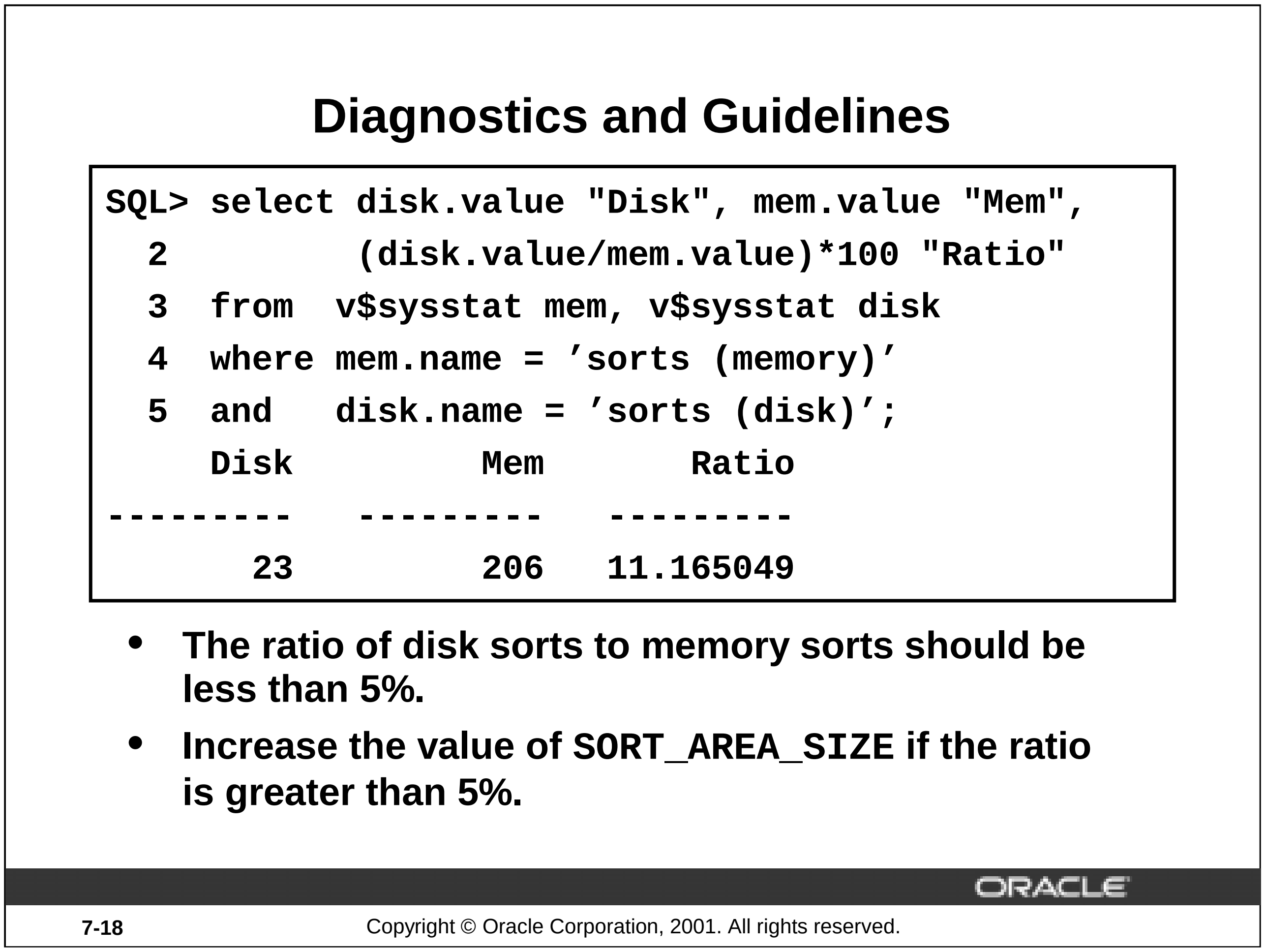
```
  SQL> select * from v$sysstat where name like '%sorts%';
  STATISTIC# NAME                             CLASS     VALUE
---------------------------                   -------   ------
  161 sorts (memory)                         64        154
  162 sorts (disk)                           64        4
  163 sorts (rows)                           64        571768
```

Dynamic Views and STATSPACK Output (continued)

The STATSPACK output gives the same information. Moreover, these figures cover the period when the snapshots ran.

```
Statistic           TotalPer Transact     Per Logon     Per Second
-----------     --------- ----------     ----------    -----------
sorts (disk)            4        .02            .41            .01
sorts (memory)        154        .27           5.77            .12
sorts (rows)       571768      39.62         862.59          18.19
```

The V$SORT_SEGMENT and V$SORT_USAGE views display information about the temporary segments used and the users who used them.



Diagnostics and Guidelines

```
SQL> select disk.value "Disk", mem.value "Mem",
  2         (disk.value/mem.value)*100 "Ratio"
  3  from  v$sysstat mem, v$sysstat disk
  4  where mem.name = ’sorts (memory)’
  5  and   disk.name = ’sorts (disk)’;
     Disk         Mem       Ratio
---------   ---------   ---------
       23         206   11.165049
```

- **The ratio of disk sorts to memory sorts should be less than 5%.**
- **Increase the value of `SORT_AREA_SIZE` if the ratio is greater than 5%.**

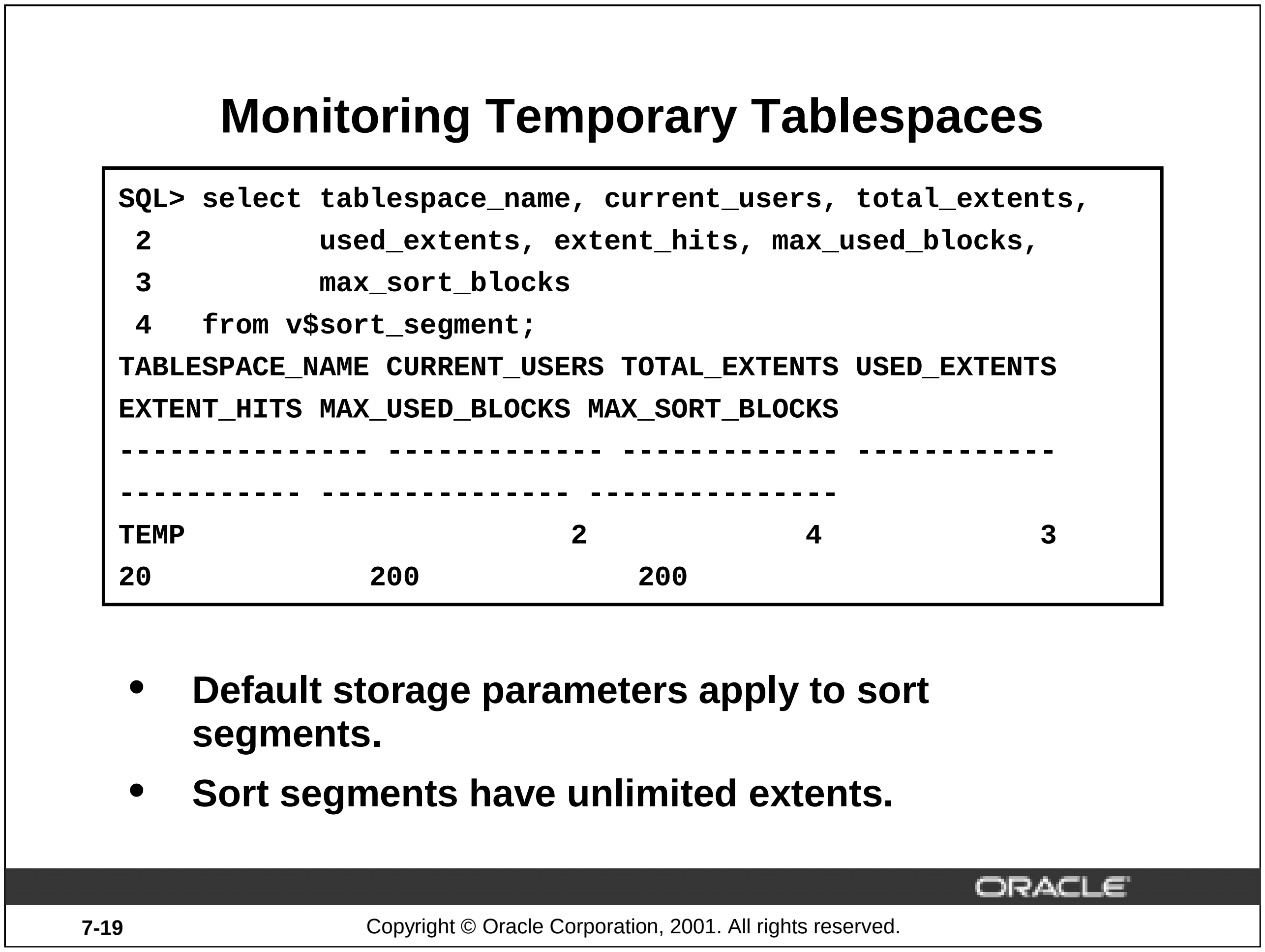
Ratio

The ratio of sorts (disk) to sorts (memory) should be less than 5%. If the ratio indicates a high number of sorts going to disk, increase the value of `SORT_AREA_SIZE.` This increases the size of each run and decreases the total number of runs and merges.

Performance Trade-Offs for Large Sort Areas

Increasing the size of the sort area causes each server process that sorts to allocate more memory. It may affect operating system memory allocation and induce paging and swapping.

If you increase sort area size, consider decreasing the retained size of the sort area, or the size to which the server reduces the sort area if its data is not expected to be referenced soon. A smaller retained sort area reduces memory usage but causes additional I/O to write and read data to and from temporary segments on disk.



Monitoring Temporary Tablespaces

```
SQL> select tablespace_name, current_users, total_extents,
 2         used_extents, extent_hits, max_used_blocks,
 3         max_sort_blocks
 4   from v$sort_segment;
TABLESPACE_NAME CURRENT_USERS TOTAL_EXTENTS USED_EXTENTS
EXTENT_HITS MAX_USED_BLOCKS MAX_SORT_BLOCKS
--------------- ------------- ------------- ------------
----------- --------------- ---------------
TEMP                       2             4            3
20             200              200
```

- **Default storage parameters apply to sort segments.**
- **Sort segments have unlimited extents.**

The `V$SORT_SEGMENT` View

This view contains information about every sort segment of the TEMPORARY tablespaces in the instance.

Column	**Description**
`CURRENT_USERS`	Number of active users
`TOTAL_EXTENTS`	Total number of extents
`USED_EXTENTS`	Extents currently allocated to sorts
`EXTENT_HITS`	Number of times an unused extent was found in the pool
`MAX_USED_BLOCKS`	Maximum number of used blocks
`MAX_SORT_BLOCKS`	Maximum number of blocks used by an individual sort

Temporary Tablespace Configuration

Default storage parameters for the temporary tablespace apply to sort segments, except that they have unlimited extents—whatever value `MAXEXTENTS` is set to.

Temporary Tablespace Configuration

- **Set appropriate storage values.**
- **Set up different temporary tablespaces based on sorting needs.**

```
SQL> SELECT session_num, tablespace, extents, blocks
 2   FROM v$sort_usage;
SESSION_NUM  TABLESPACE            EXTENTS    BLOCKS
-----------  --------------------- -------- --------
         16  TEMP                         4      200
```

- **Stripe temporary tablespaces.**
- **Use `v$tempfile` and `dba_temp_files` for information on temporary files.**

ORACLE

Guidelines

Storage Parameters

Because sorts are performed in memory if they are smaller than `SORT_AREA_SIZE`, you should consider this value when setting extent sizes:

- Select `INITIAL` and `NEXT` values as integer multiples of `SORT_AREA_SIZE`, allowing an extra block for the segment header.
- Set `PCTINCREASE` to 0.

Different Temporary Tablespaces

To define the sort space needed by the users, and to obtain information on the currently active disk sorts in the instance:

```
SQL> SELECT username, tablespace, contents, extents, blocks
    2> FROM v$sort_usage ;
```

Guidelines (continued)

Different Temporary Tablespaces (continued)

```
USERNAMETABLESPACE    CONTENTS    EXTENTS    BLOCKS
--------------------------------------------------------------
HR           TEMP     TEMPORARY         20      1000
OE           TEMP     TEMPORARY          2       100
```

The `USERNAME` column in `V$SORT_USAGE` always shows the user querying this view. The user performing the sort is the username from the `V$SESSION` view. The user `stat` requiring a large amount of disk space should be assigned another temporary tablespace with larger extent size.

Striping

The temporary tablespace should be striped over many disks. If the temporary tablespace is striped over only two disks with a maximum of 50 I/Os per second each, then you can only do 100 I/Os per second. This restriction may become a problem, making sort operations take a very long time. You can speed up sorts fivefold by striping the temporary tablespace over ten disks, thus allowing 500 I/Os per second.

Datafiles

You also use different views for viewing information about temporary files than you would for datafiles. The `V$TEMPFILE` and `DBA_TEMP_FILES` views are analogous to the `V$DATAFILE` and `DBA_DATA_FILES` views.

Summary

In this lesson, you should have learned how to:

- **Describe how sorts are performed**
- **Identify the SQL operations that require sorts**
- **List ways to reduce total sorts and disk sorts**
- **Determine the number of sorts performed in memory**
- **Set old and new sort parameters**
- **Differentiate between disk and memory sorts**

Quick Reference

Context	**Reference**

Initialization parameters	`SORT_AREA_SIZE`
	`SORT_AREA_RETAINED_SIZE`
	`SORT_MULTIBLOCK_READ_COUNT`

Dynamic performance views	`V$SYSSTAT`
	`V$SORT_SEGMENT`
	`V$SORT_USAGE`
	`V$SESSION`

Data dictionary views	None
Commands	None
Packaged procedures and functions	None
Scripts	None
Oracle Diagnostics Pack	Performance Manager

Practice 7

1. Connect as system/manager and query the V$SYSSTAT view, and record the value for sorts (memory) and sorts (disk). If the ratio of Disk to Memory sorts is greater than 5% then increase the sort area available.

   Note: The statistics collected from v$sysstat are collected from startup. If you need to get accurate statistics per statement, you must record statistics from before the statement has run and again afterwards. Subtracting to two values will give the statistics for the statement.

2. Connect as user sh/sh. In order to ensure that some sorts go to disk run the command "alter session set sort_area_size = 512;". Then execute the SQL script ($HOME/STUDENT/LABS/lab07_02.sql) that will force sorts to disk.

   Note: If this script fails due to a lack of free space in the TEMP tablespace. Resize the temporary tablespace.

3. Connect as system/manger and query the columns TABLESPACE_NAME, CURRENT_USERS, USED_EXTENTS and FREE_EXTENTS from the V$SORT_SEGMENT view. The columns USED_EXTENTS, and FREE_EXTENTS are useful in monitoring the usage of the TEMPORARY tablespace.

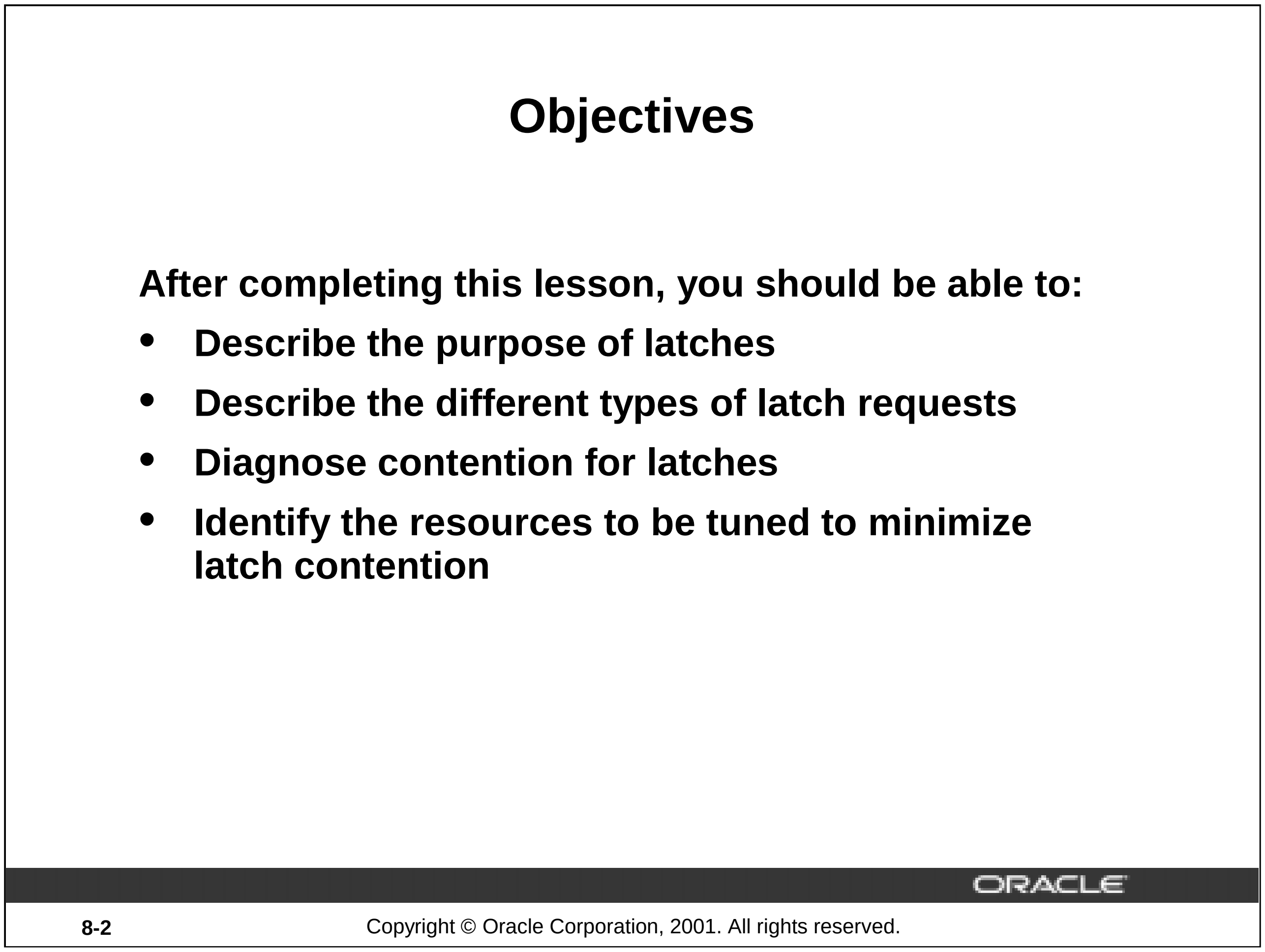
   Note: If this statement returns no rows, it means that all sort operations since startup have completed in memory.

4. To decrease the sorts number of sorts going to a temporary tablespace, increase the value of the parameter SORT_AREA_SIZE to 512000 using the "alter session" command.

5. Connect as system/manager and configure the new parameters for PGA memory allocation using the "alter system" command. Use the values AUTO for WORKAREA_SIZE_POLICY and 10M for PGA_AGGREGATE_TARGET)

8

Diagnosing Contention for Latches

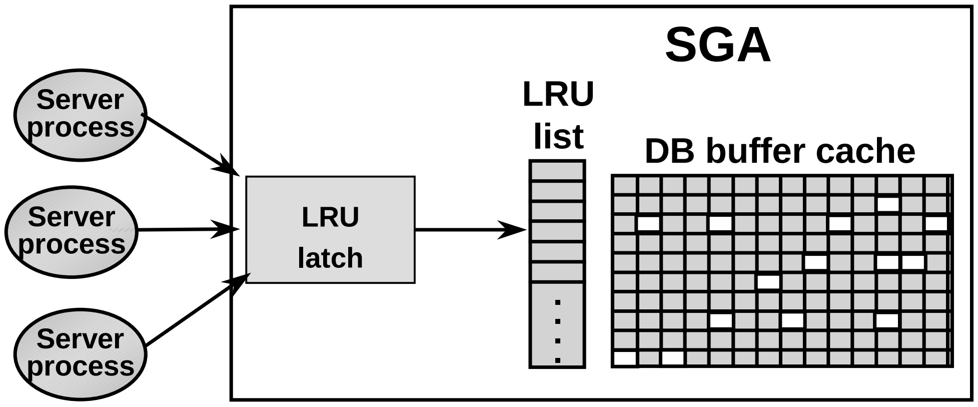


Objectives

After completing this lesson, you should be able to:

- Describe the purpose of latches
- Describe the different types of latch requests
- Diagnose contention for latches
- Identify the resources to be tuned to minimize latch contention

Latches: Overview



What are Latches

Latches are simple, low-level serialization mechanisms to protect shared data structures in the system global area (SGA). For example, latches protect the list of users currently accessing the database, and protect the data structures describing the blocks in the buffer cache. A server or background process must acquire the accompanying latch to start manipulating or looking at a shared data structure and must release the accompanying latch when finished. The implementation of latches is operating system and platform dependent, particularly in regard to whether and how long a process will wait for a latch.

Tuning Latches

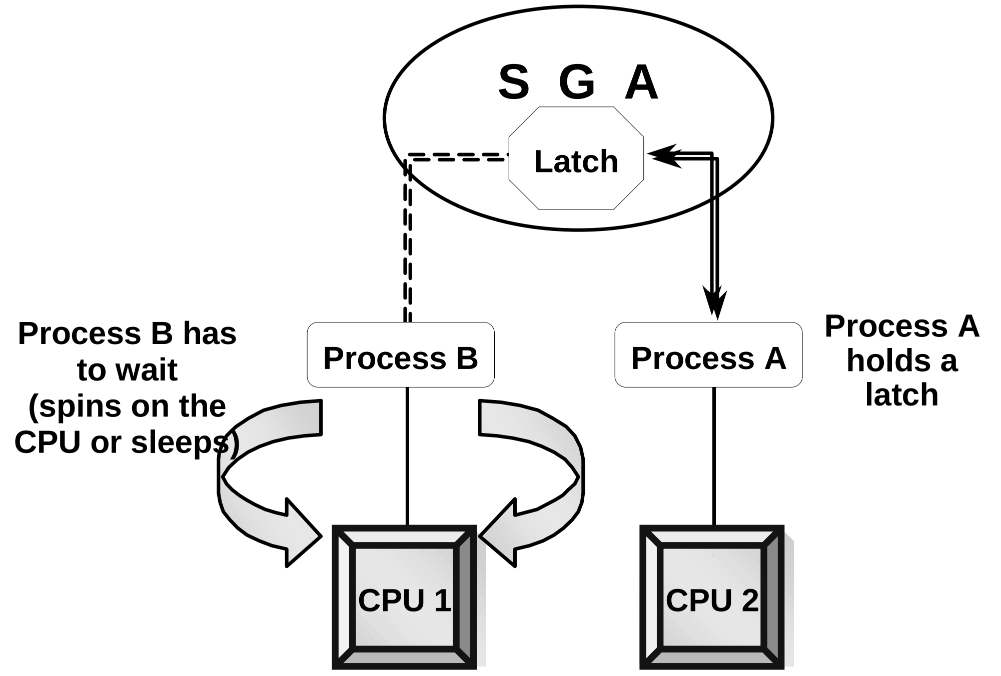
You *do not* tune latches. If you see latch contention, it is a symptom of a part of SGA experiencing abnormal resource usage. Latches control access with certain assumptions, for example, a cursor is parsed once and executed many times. To fix the problem, examine the resource usage for the parts of SGA experiencing contention. Merely looking at V$LATCH does not address the problem.

In summary, latches are light weight locks protecting internal data structures. In Oracle9*i*, latches are inaccessible to users, yet latch contention statistics can serve as a diagnostic tool to identify what resources should be tuned in order to optimize performance.

Purpose of Latches

- To serialize access:
  - Protect data structures within the SGA
  - Protect shared memory allocations
- To serialize execution:
  - Prevent simultaneous execution of certain *critical* pieces of code
  - Prevent corruptions

Waiting for a Latch

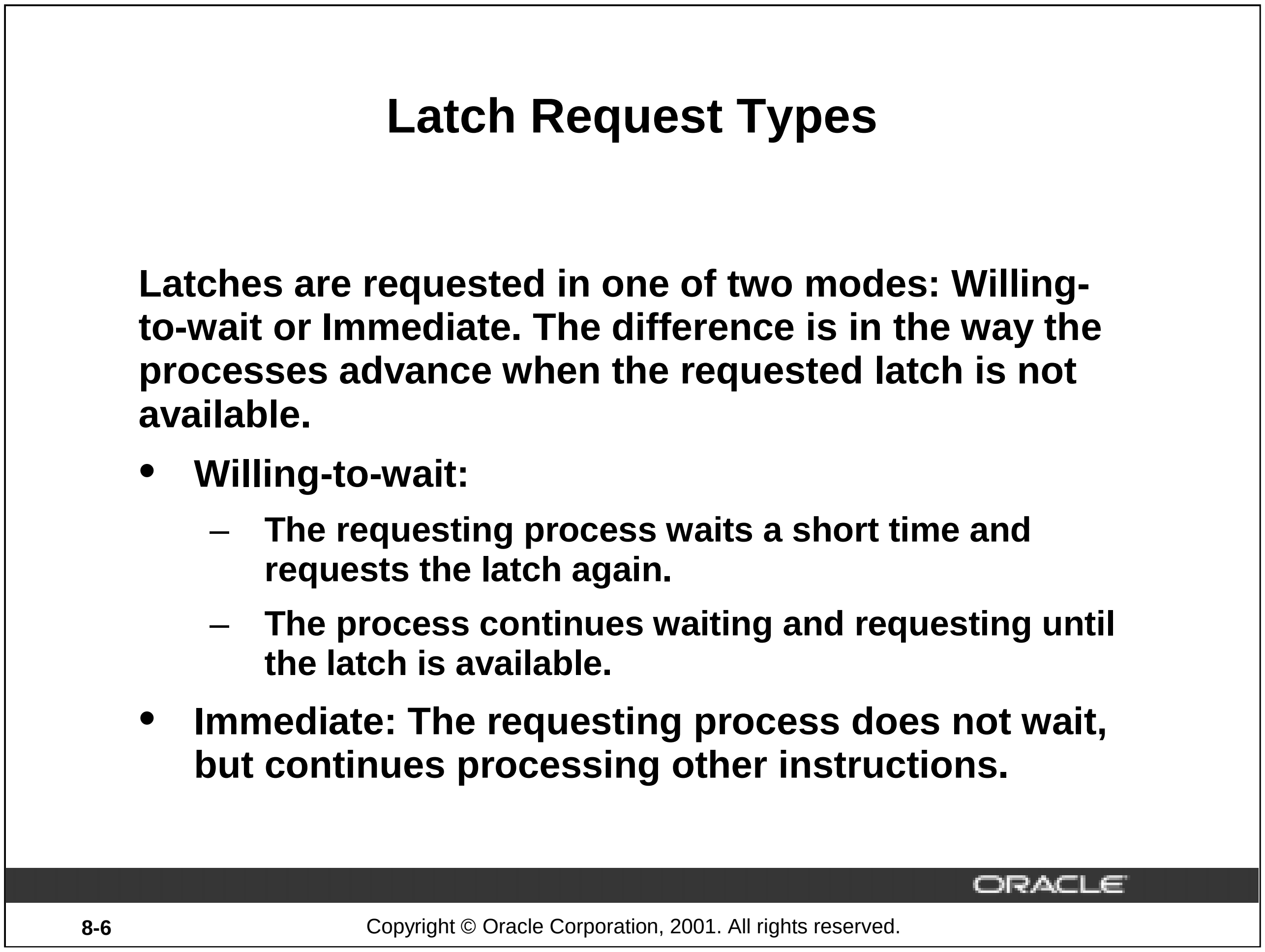


Waiting for a Latch

Although the exact implementation is operating system and platform specific, latches are normally implemented as a single memory location that has a value of zero if the latch is free, or non-zero if it is already acquired.

On single-CPU systems, if a required latch is already held by another process, the process requesting the latch will release the CPU and go to sleep for a brief period before trying again. A sleep corresponds to a wait for the latch free wait event.

On a multi-CPU system it is possible that the process holding the latch is running on another CPU and so might potentially release the latch in the next few instructions. Therefore, on multi-CPU systems the requesting process holds the CPU and spins (counts up to a specific number), then tries to acquire the latch again; and if still not available, spins again. The number of spins and the spin time is operating system and platform specific. If after the number of spins the latch is still not available, the process releases the CPU and goes to sleep, like in the single-CPU case. However, since latches are normally held for brief periods of time (order of microseconds), a successful spin in the multi-CPU case avoids a costly context switch at the expense of holding on to the CPU.



Latch Request Types

Latches are requested in one of two modes: Willing-to-wait or Immediate. The difference is in the way the processes advance when the requested latch is not available.

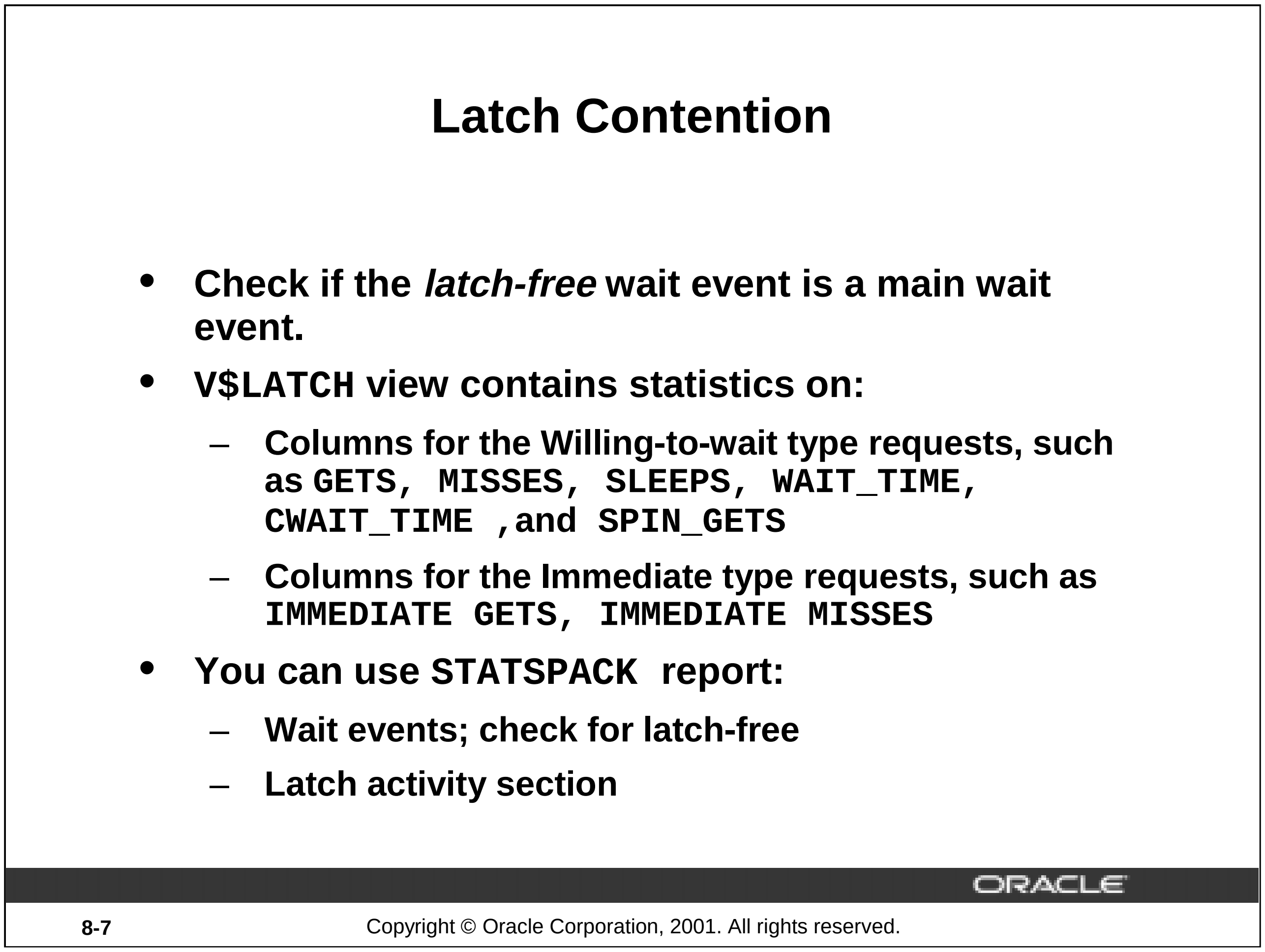
- **Willing-to-wait:**
  - **The requesting process waits a short time and requests the latch again.**
  - **The process continues waiting and requesting until the latch is available.**
- **Immediate: The requesting process does not wait, but continues processing other instructions.**

8-6

Latch Requests

Processes request latches in one of two modes: willing-to-wait or immediate. The essential difference between the two request types is the way the processes proceed when the requested latch is not available.

- Willing-to-wait: If the latch requested with a willing-to-wait request is not available, the requesting process waits a short time and requests the latch again. The process continues waiting and requesting until the latch is available. This is the usual way of processing.
- Immediate: If the latch requested with an immediate request is not available, the requesting process does not wait, but continues processing other instructions. For example, when the PMON process attempts to clean up an abnormally terminated process, it finds that the latch required to access the structure is not available. PMON continues with subsequent instructions rather than waiting for the latch to be freed.



Latch Contention

- Check if the *latch-free* wait event is a main wait event.
- `V$LATCH` view contains statistics on:
  – Columns for the Willing-to-wait type requests, such as `GETS, MISSES, SLEEPS, WAIT_TIME, CWAIT_TIME` ,and `SPIN_GETS`
  – Columns for the Immediate type requests, such as `IMMEDIATE GETS, IMMEDIATE MISSES`
- You can use `STATSPACK` report:
  – Wait events; check for latch-free
  – Latch activity section

8-7

Latch Contention

If multiple processes are seeking the same latch, any process can acquire the latch, depending on the time of its re-seek and the release of the latch. However, latches are held by atomic instructions and hence are obtained and released very quickly. The goal is to minimize the contention for latches among processes. Like all resources, available latches are finite.

Diagnostics

As a first step, check if the wait event `latch free` has a high wait time. If it does, review statistics from the `V$LATCH` view. The three TIME columns are added in 9*i*.

The following columns in the `V$LATCH` view reflect willing-to-wait requests:

- `gets`: Number of successful willing-to-wait requests for a latch
- `misses`: Number of times an initial willing-to-wait request was unsuccessful
- `sleeps`: Number of times a process waited after an initial willing-to-wait request
- `wait_time`: Number of milliseconds waited after willing-to-wait request
- `cwait_time`: A measure of the cumulative wait time including the time spent spinning and sleeping, the overhead of context switches due to OS time slicing and page faults and interrupts
- `spin_gets`: Gets that missed first try but succeeded after spinning

Latch Contention (continued)

The following columns in the V$LATCH view reflect immediate requests:

- `immediate_gets`: Number of successful immediate requests for each latch.
- `immediate_misses`: Number of unsuccessful immediate requests for each latch.

STATSPACK report

The report produced by `STATSPACK` contains some sections meant for diagnosing latch contention. First however, you should look at the `Top 5 Wait Events` section. If latch free event is one of them it may be worth while to diagnose further to find which latches are involved in contention. Following are sections from a `STATSPACK` report.

```
Top 5 Wait Events
~~~~~~~~~~~~~~~~~
Events                   Waits        Wait Time    % Total
                                      (cs)         Wt Time
---------------------------           ---------    --------- --------
enqueue                    482,210    1,333,260    36.53
latch free               1,000,676      985,646    27.01
buffer busy waits          736,524      745,857    20.44
log file sync              849,791      418,009    11.45
log file parallel write    533,563      132,524     3.63
        -----------------------------------


Latch Activity for DB: ED31  Instance: ed31  Snaps: 1 -2
->"Get Requests", "Pct Get Miss" and "Avg Slps/Miss" are statistics for
  willing-to-wait latch get requests
->"NoWait Requests", "Pct NoWait Miss" are for no-wait latch get requests
->"Pct Misses" for both should be very close to 0.0
-> ordered by Wait Time desc, Avg Slps/Miss, Pct NoWait Miss desc


                                           Pct    Avg           Pct
                              Get          Get   Slps   NoWait NoWait
Latch                       Requests      Miss  /Miss  Requests   Miss
------------------------            ------------ -----   ------ -------
    --------

…
cache buffers chains     142,028,625 0.3     0.4  1,193,834 0.7
cache buffers lru chain    8,379,760 0.1     0.7    414,979 0.1
…
library cache             36,870,207 2.1     0.7          0 0.4
…
Redo allocation           10,478,825 0.9     0.3          0
    …
Row cache objects         27,905,682 0.3     0.1          0
…
```

Reducing Contention for Latches

Generally, the DBA should not attempt to tune latches. However, the following steps are useful:

- **Investigate further, depending on the latch that is in contention.**
- **Consider tuning the application if the contention is mainly for shared pool and library cache.**
- **If further investigation suggests it, size shared pool and buffer cache appropriately.**

8-9

Reducing Contention

In Oracle9*i*, you should not be attempting to tune the number of latches. Oracle9*i* automatically calculates the number of latches required, based on the environment defined by the initialization parameters and the operating system level parameters.

Contention for latches is a symptom of a performance problem. The type of problem is indicated by the latch contended for.

Frequently, the most effective way to reduce latch contention is to modify the application behavior. If you observe, based on other investigation of cache hit ratios, and so on, that the SGA is inappropriately configured, you may consider changing buffer cache or shared pool sizes.

Important Latches for the DBA

- `Shared pool`: Protects memory allocations in the shared pool
- `Library cache`: Used to locate matching SQL in the shared pool
- `Cache buffers LRU chain`: Protects the LRU list of cache buffers
- `Cache buffers chains`: Needed when searching for data blocks cached in the SGA
- `Redo allocation`: Manages the allocation of redo space in the log buffer for redo entries
- `Redo copy`: Used to write redo records into the redo log buffer

Significant Latches

Contention for the following latches may be significant in your database:

- `shared pool latch`, `and library cache latch`:

  Contention for these latches indicates that SQL, PL/SQL statements are not being reused, possibly because the statements are not using bind variables, or the cursor cache is insufficient. To alleviate the problem, consider tuning the shared pool or the application.

  Using Oracle Shared Server without a large pool may lead to contention on the Shared Pool Latch. This can be resolved by creating the large pool, or by reverting to direct connections.

- `cache buffers lru chain latch`:

  This latch is required when dirty blocks are written to the disk or when a server process is searching for blocks to write to. Contention for this latch indicates excessive buffer cache throughput, such as many cache-based sorts, inefficient SQL that accesses incorrect indexes iteratively (large index range scans), or many full table scans. It is also possible that the database writer is unable to keep pace with the rate of changes to data blocks, forcing the foreground process to wait longer holding the latch while looking for a free buffer. To overcome the problem, consider tuning the buffer cache or the database writer operation.

Significant Latches (continued)

- `cache buffers chains latch`:

  This latch is needed when user processes try to locate a data block in the buffer cache. The contention for this latch indicates some specific blocks (hot blocks) are accessed repeatedly.

The following Oracle8*i* `init.ora` parameters related to latches are now obsolete:

`DB_BLOCK_LRU_LATCHES` specifies the maximum number of LRU latch sets, i.e., cache buffers LRU chain and cache buffer chains. The buffers of a buffer pool are equally divided among the working LRU latch sets of the buffer pool so that each buffer is protected by one LRU latch. Normally, the more latches you specify, the less contention exists for those latches. However, too many latches may result in small LRU lists, potentially reducing the cache life of a database block.

The maximum of (`CPU_COUNT x 2 x 3`) ensures that the number of latches does not exceed twice the product of the number of CPUs and the number of buffer pools. Typically you should set this parameter to the number of CPUs or a multiple of that number. Each working set is handled entirely by one database writer (DBW*n*) process. Therefore, if multiple DBW*n* processes are running, the number of LRU latches should be greater than or equal to the number of DBW*n* processes. To balance the load evenly between the DBW*n* processes, the number of LRU latches in each buffer pool should be a multiple of the number of DBW*n* processes.

If you do not set this parameter, Oracle Server uses the value `CPU_COUNT/2`. This value is usually adequate. Increase this value only if misses are higher than 3%, as calculated from values in `V$LATCH`. When you increase the value, Oracle Server decides whether to use this value or reduce it based on a number of internal checks.

Shared Pool and Library Cache Latches

Contention for shared pool latch and library cache latch indicates one or more of the following:

- **Unshared SQL**
- **Reparsed sharable SQL**
- **Insufficiently sized library cache**

Shared Pool and Library Cache Latch Contention:

A main cause of shared pool or library cache latch contention is unnecessary parsing. The methods to overcome this problem are also discussed in the "Tuning Shared Pool" lesson. There are a number of techniques that can be used to identify unnecessary parsing and a number of types of unnecessary parsing:

- Unshared SQL: Identify similar SQL statements that could be shared if literals were replaced with bind variables. You could inspect SQL statements that have only one execution to see if they are similar. If you specify the sort by uppercase, it is easier to notice similar SQL statements.

```
SELECT sql_text
FROM V$SQLAREA
WHERE executions = 1
ORDER BY UPPER(sql_text);
```

- Reparsed Sharable SQL: Check the `V$SQLAREA` view to find if there are avoidable reparsing.

```
SELECT SQL_TEXT, PARSE_CALLS, EXECUTIONS
FROM V$SQLAREA
ORDER BY PARSE_CALLS;
```

.

Summary

In this lesson, you should have learned how to:

- Describe latches and their usage
- Diagnose contention for latches
- Identify the resources to be tuned to minimize contention for latches

9

Tuning Rollback Segments

Objectives

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

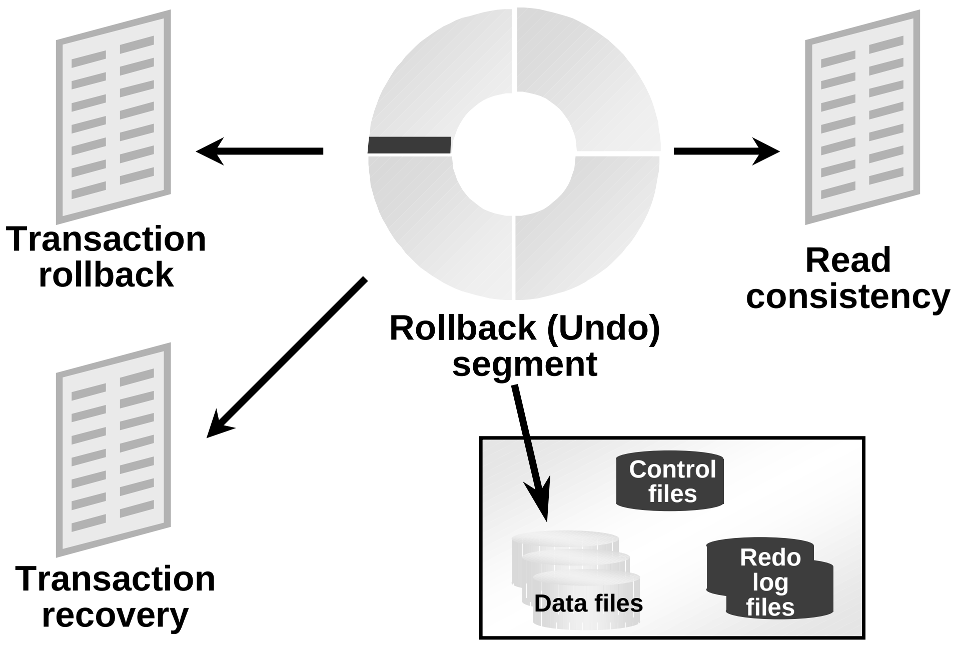
- Use the dynamic performance views to check rollback segment performance
- Reconfigure and monitor rollback segments
- Define the number and sizes of rollback segments
- Appropriately allocate rollback segments to transactions
- Describe the concept of automatic undo management
- Create and maintain the automatic managed undo tablespace

Objectives

Good rollback segment configuration is crucial to a well-tuned Oracle database. This lesson helps you to recognize and solve problems arising from inappropriate numbers or sizes of rollback segments.

The automatic undo management feature has been introduced in Oracle9*i*. This lesson helps you understand the automatic undo management feature. You will also learn to configure automatic undo management in an Oracle9*i* database.

Rollback Segments: Usage



ORACLE

9-3 Copyright © Oracle Corporation, 2001. All rights reserved.

Transaction Rollback

When a transaction makes changes to a row in a table, the old image is saved in a rollback segment, also called an *undo segment*. If the transaction is rolled back, the value in the rollback segment is written back to the row, restoring the original value.

Transaction Recovery

If the instance fails when transactions are in progress, the Oracle server rolls the uncommitted changes back when the database is opened again. This rollback is known as *transaction recovery*, and is possible only if changes made to the rollback segment are also protected by the redo log files.

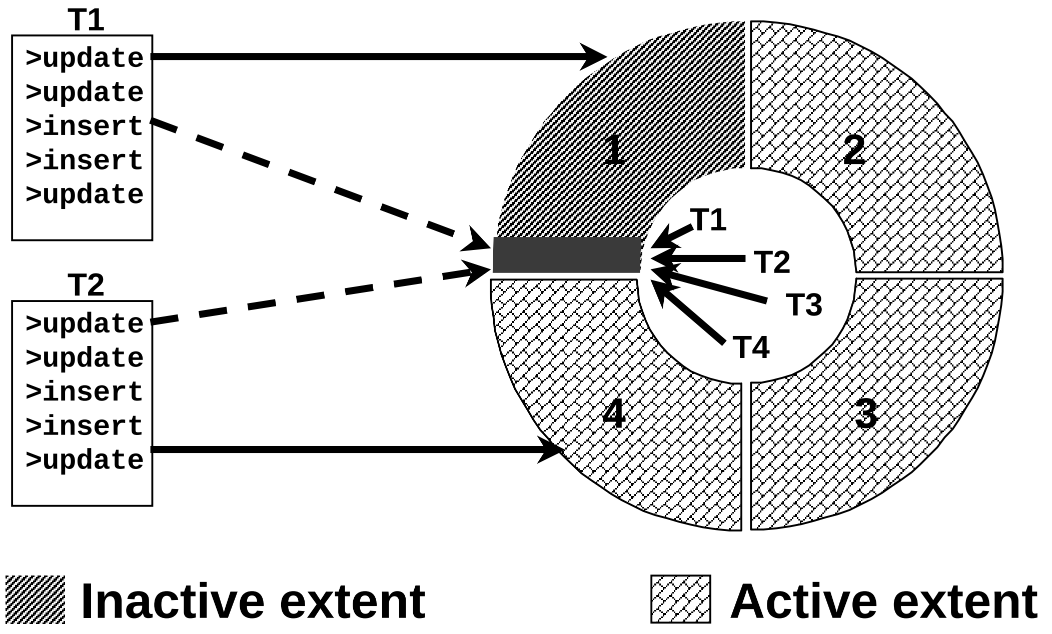
Read Consistency

Read consistency consists of the following conditions:

- When a user’s transactions are in progress, other sessions in the database should not see any uncommitted changes.
- A statement should not see any changes that were committed after the statement commenced execution.

The old values in the rollback segments, also referred to as undo information, are used to provide the read-consistent image.

Rollback Segment Activity



Active and Inactive Extents

Transactions use extents of a rollback segment in an ordered, circular fashion, moving from one extent to the next after the current extent is full. A transaction writes a record to the current location in the rollback segment and advances the current pointer by the size of the record.

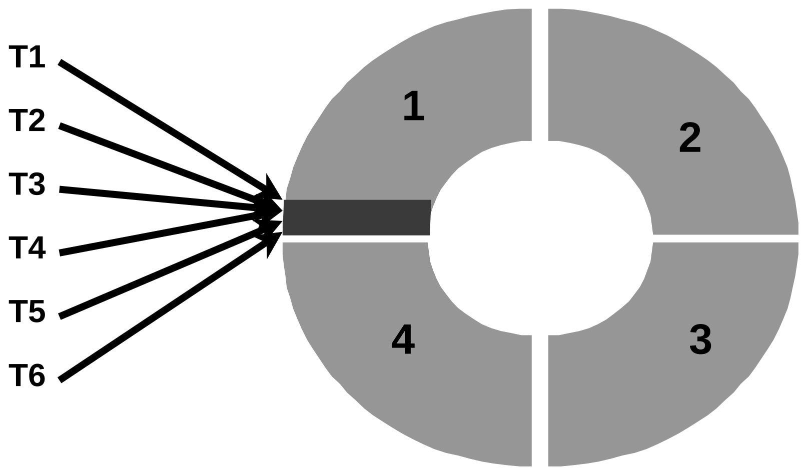
Note: More than one transaction can write to the same extent of a rollback segment. Each rollback segment block contains information from only one transaction.

Rollback Segment Activity

Writing to rollback segments requires that the corresponding undo data is available in the database buffer cache. To maintain large amounts of undo information, the buffer cache should be quite large, or there is a higher number of physical I/Os. This manifests as a performance issue in situations where numerous OLTP transactions and long running DSS queries are performed simultaneously.

Rollback Segment Header Activity

- **Rollback segment headers contain entries for their respective transactions.**
- **Every transaction must have update access.**



Rollback Segment Header Activity

The Oracle server keeps a transaction table in the header of every rollback segment.

The rollback segment header activity controls the writing of changed data blocks to the rollback segments. Because the rollback segment header is a data block and it is frequently modified, the rollback segment header block remains in the data block buffer cache for long periods of time. Therefore, accesses to the rollback segment header block increase the hit ratio for the application, even though it is not related to the data blocks.

The Impact of Rollback Segment Header Activity

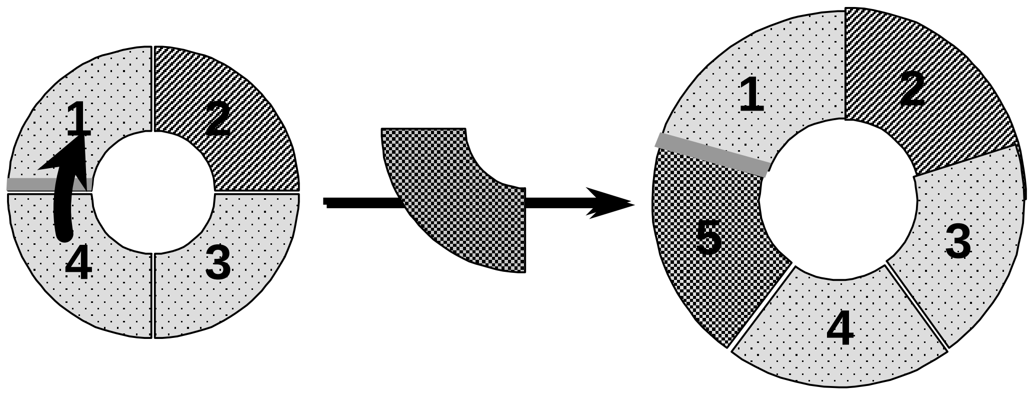
The impact of the rollback segment header activity on the cache hit ratio is important for OLTP systems that feature many small transactions.

Every transaction must have update access to the transaction table for its rollback segment. You need enough rollback segments to prevent transactions from contending for the transaction table.

If you underestimate the number of rollback segments needed, performance is degraded and transactions may generate errors. If you overestimate, you use unnecessary space.

When using the automatic undo management feature, Oracle server automatically manages the number of undo segments, thus relieving the DBA of this burden.

Growth of Rollback Segments



▨ **Active extent**

▨ **Inactive extent**

▩ **New extent**

Growth of Rollback Segments

The pointer or the head of the rollback segment moves to the next extent when the current extent is full. When the last extent that is currently available is full, the pointer can move back to the beginning of the first extent only if that extent is free. The pointer cannot skip over an extent and move to the second or any other extent.

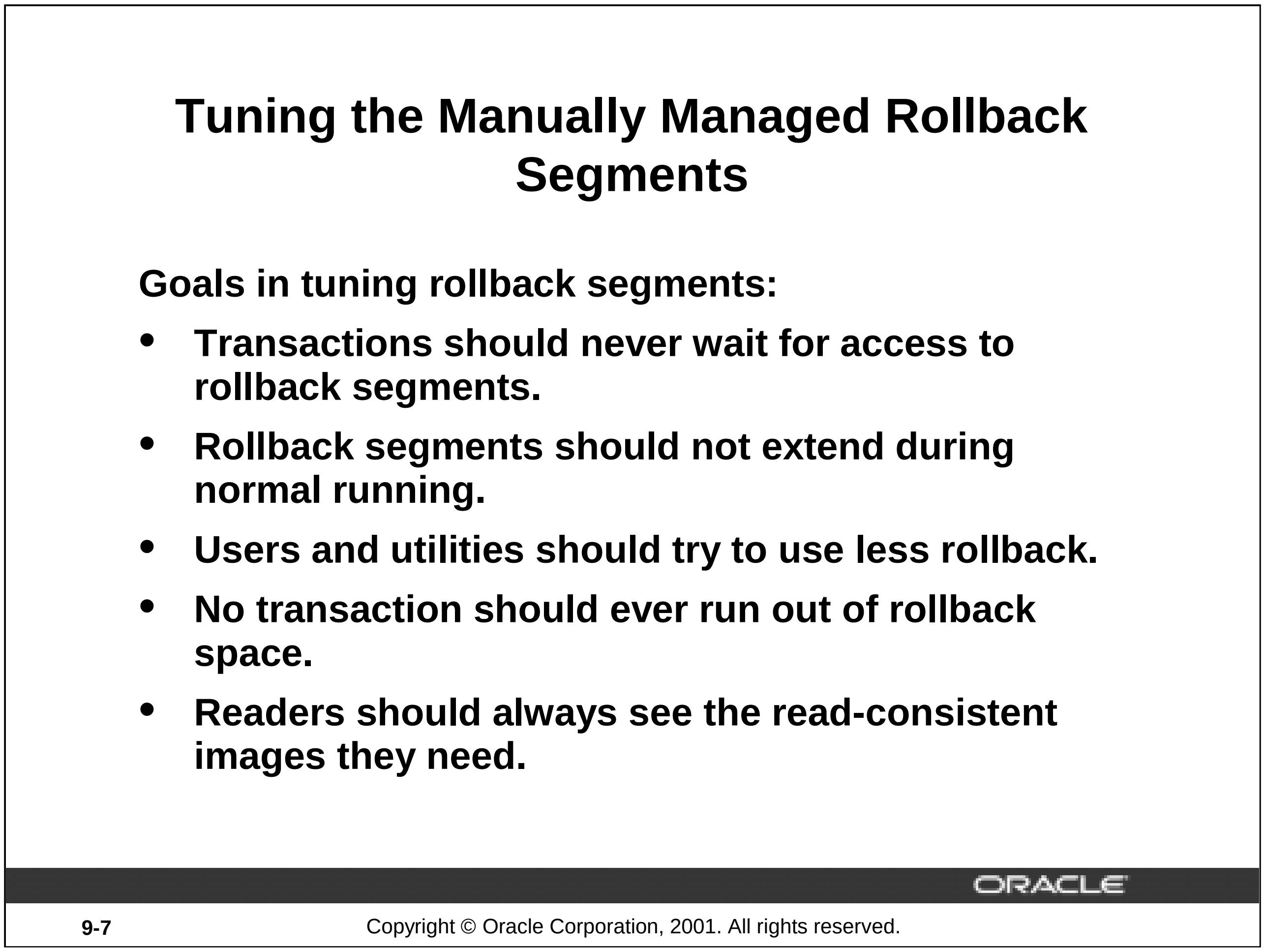
If the first extent is being used, the transaction allocates an additional extent for the rollback segment. This is called an *extend*. Similarly, if the head tries to move into an active extent, the rollback segment allocates an additional extent.

The Impact of Rollback Segment Extending

Rollback segments should not be extended during normal running. To prevent this, rollback segments must have enough extents to hold the rollback entries for the transactions.

As with other objects, you should avoid dynamic space management.

If you underestimate the size of rollback segments, performance is degraded and transactions may generate errors. If you overestimate, you use unnecessary space.



Tuning the Manually Managed Rollback Segments

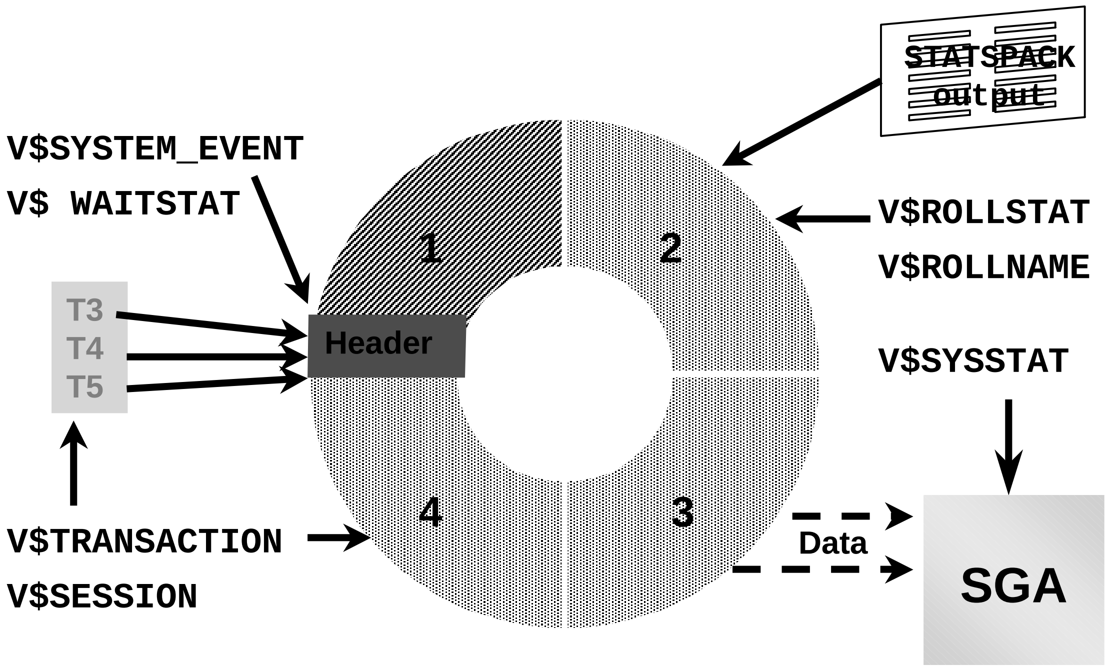
Goals in tuning rollback segments:

- **Transactions should never wait for access to rollback segments.**
- **Rollback segments should not extend during normal running.**
- **Users and utilities should try to use less rollback.**
- **No transaction should ever run out of rollback space.**
- **Readers should always see the read-consistent images they need.**

Tuning Goals

- Transactions should never wait for access to rollback segments. This requires you to have enough rollback segments.
- Rollback segments should not extend during normal running. This requires:
  - An appropriate number of extents per segment
  - The correct sizing of the extents
  - The appropriate number of rollback segments
  - A better use of utilities that can use less rollback
- No transaction, however large or exceptional, should ever run out of rollback space. This means that:
  - Rollback segments should be resized correctly.
  - Large transactions should be split into smaller transactions.
- Readers should always be able to see the read-consistent images they need. This requires the appropriate:
  - Number of rollback segments
  - Sizing of rollback segments

Diagnostic Tools



Dynamic Views to Monitor Rollback Activity

- `V$ROLLNAME`: displays the name and number of the online rollback segments
- `V$ROLLSTAT`: displays statistics of the activity for each online rollback segment:
  - Number of waits on the header transaction table
  - Volume of data written by the transactions
- `V$SYSTEM_EVENT`: The Undo Segment Tx Slot event shows waits for transaction slots and therefore contention on rollback segment headers.
- `V$WAITSTAT`: displays the cumulative statistics of waits on header blocks and data blocks of all rollback segments
- `V$SYSSTAT`: displays the number of consistent and data block gets. You can calculate the number of waits with the total number of requests for data.
- `V$TRANSACTION`: displays the current transactions using rollback segments and therefore the volume of required space

Except for `V$ROLLNAME`, all of these views use the undo segment number (USN) as the identifier for rollback. So when you need to get the name of the rollback segment, join the `V$ROLLNAME` on USN column.

Diagnosing Contention for Manual Rollback Segment Header

```
SQL> SELECT class, count FROM v$waitstat
  2  WHERE class LIKE ‘%undo%’;
or
SQL> SELECT sum(value) FROM v$sysstat
  2  WHERE name IN (‘db block gets‘, 'consistent gets‘);
or
SQL> SELECT sum(waits)* 100 /sum(gets) "Ratio",
  2  sum(waits) "Waits", sum(gets) "Gets"
  3  FROM   v$rollstat;
```

- **The number of waits for any class should be less than 1% of the total number of requests.**
- **If not, create more rollback segments.**

Diagnosing Contention for Rollback Segment Header

A nonzero value in the following indicates contention for rollback segments:

- waits column of the V$ROLLSTAT view
- undo header row of the V$WAITSTAT view
- Undo Segment Tx Slot event of the V$SYSTEM_EVENT view

The following statement queries the V$WAITSTAT view to look for contention on the rollback segment:

```
SQL> select class, count from v$waitstat
  2  where class like ‘%undo%‘;
```

Diagnosing Contention for Rollback Segment Headers (continued)

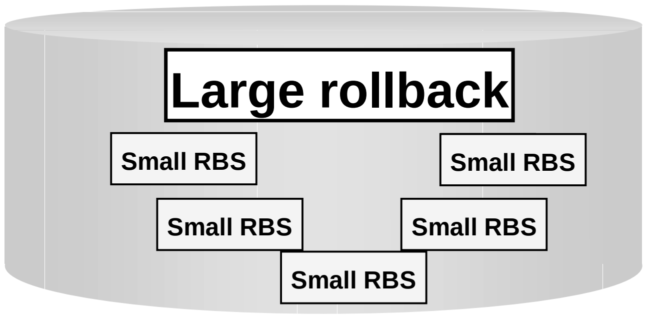
The rollback and undo related information from the `STATSPACK` are located mainly in the Rollback Segment Stats section. Following is an example of the Rollback Segment Stats section:

```
  Rollback Segment Stats for DB: ED31  Instance: ed31  Snaps: 1 –2
  ->A high value for "Pct Waits" suggests more rollback segments may be
  required
       Trans Table      Pct   Undo Bytes
RBS No     Gets       Waits     Written        Wraps  Shrinks  Extends
------ ------------ ------- --------------- -------- -------- --------
     0          5.0    0.00               0        0        0        0
     1         66.0    0.00           5,636        0        0        0
     2        439.0    0.00         358,772        5        0        0
     3         50.0    0.00           6,314        0        0        0
     4         53.0    0.00           7,004        0        0        0
        -------------------------------------------------------------
```

Guideline

When you observe contention for rollback segments, you should investigate further the cause of contention to determine if the mere addition of rollback segments would alleviate the problem, or if it is necessary to configure the tablespaces to manage the I/O.

Guidelines: Number of Manual Rollback Segments (RBSs)



- **OLTP: One RBS for four transactions**
- **Batch: One rollback segment for each concurrent job**

```
SQL> SET TRANSACTION USE ROLLBACK SEGMENT large_rbs;
```

OLTP Transactions

- OLTP applications are characterized by frequent concurrent transactions, each of which modifies a small amount of data. Assign small rollback segments to OLTP transactions.
- The reasonable rule of thumb is one rollback segment for every four concurrent transactions.

Long Batch Transactions

Assign large rollback segments to transactions that modify large amounts of data. Such transactions generate large rollback entries. If a rollback entry does not fit into a rollback segment, the Oracle server extends the segment. Dynamic extension reduces performance and should be avoided whenever possible.

Allow for the growth of the rollback segments by creating them in large or autoextensible tablespaces, with an unlimited value for `MAXEXTENTS`.

Long Batch Transactions (continued)

- For exceptionally long transactions, you may want to assign a large rollback segment using the following syntax:

  ```
  SQL> SET TRANSACTION USE ROLLBACK SEGMENT large_rbs;
  ```

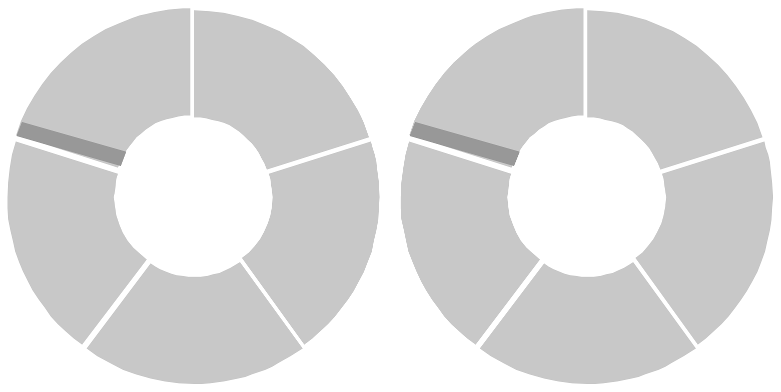
  You can also use a supplied procedure:

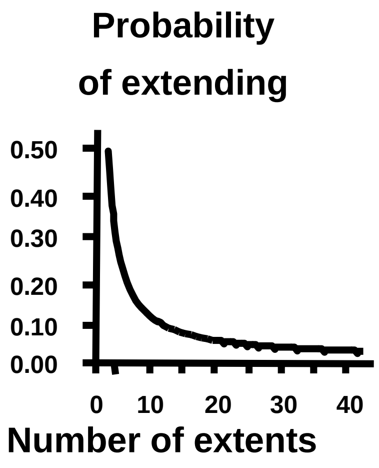
  ```
  SQL> EXECUTE
  dbms_transaction.use_rollback_segment('large_rbs');
  ```

  Remember that any commit operation, explicit or implicit, ends the transaction. This means that the command may have to be included repeatedly.

  Note: The command `SET TRANSACTION USE rollback segment` must be the first one in the transaction.

Guidelines: Sizing Manual Rollback Segments





Rollback segment 1 = Rollback segment 2

`INITIAL = NEXT = 2n = X MINEXTENTS = 20`

`OPTIMAL = 20 * X`

Storage Parameters

Setting the right size for the rollback segments is significant for performance. The aim is to reduce dynamic extension and increase the chances that undo blocks are in the buffer cache when needed.

- Choose a value for the `INITIAL` storage parameter from the list 8 KB, 16 KB, 32 KB, and 64 KB for small transactions, and 128 KB, 256 KB, 512 KB, 1 MB, 2 MB, 4 MB, and so on for larger transactions. The value should be sized large enough to prevent wrapping (an entry wraps into the next extent when it cannot find enough space in the current extent).
- Use the same value for `NEXT` as for `INITIAL`. Because `PCTINCREASE` is 0, all the other extents will have the same size as the `NEXT`.
- Make all your rollback segments the same size. Take the large rollback segments offline if they are not needed.
- Set `MINEXTENTS` to 20. This makes it unlikely that the rollback segment would need to grab another extent, because the extent that it should move into is still being used by an active transaction.

Tablespace Size

Leave enough free space in the rollback segments tablespace for a larger-than-usual transaction to be able to extend the rollback segment it is using. The `OPTIMAL` setting will later cause the extended rollback segment to shrink.

Sizing Transaction Rollback Data

- **Deletes are expensive.**
- **Inserts use minimal rollback space.**
- **Updates use rollback space, depending on the number of columns.**
- **Index maintenance adds rollback.**

```
SQL> SELECT s.username, t.used_ublk, t.start_time
  2  FROM v$transaction t, v$session s
  3  WHERE t.addr = s.taddr;
```

Transaction Statements

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:

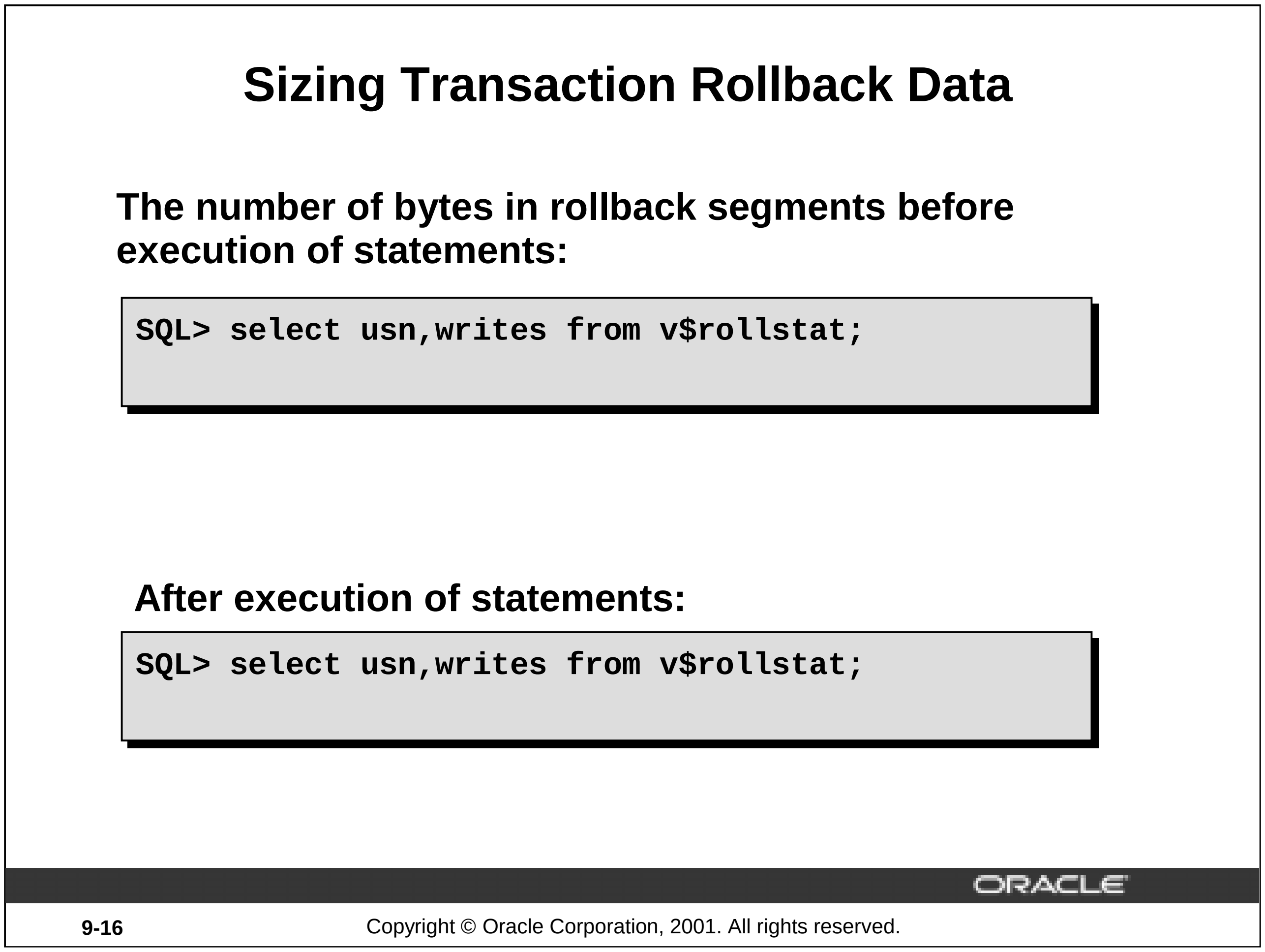
- Deletes are expensive for rollback segments; they need to store the actual row itself. If you can use `TRUNCATE` instead, performance is improved.
- Inserts use little rollback space; only the row ID is kept.
- The amount used for updates depends on how many columns are being updated.
- Indexed values generate more rollback, because the server process must change values in the index as well as in the table. For updates on indexed columns, the Oracle server records in the rollback segment the old data value, the old index value, and the new index value.

Note: Columns of the LOB datatype do not use rollback segment space for changes. They use their own segment space defined by the `PCTVERSION` clause setting: the actual data being processed.

Sizing Transaction Rollback Data Volume

Estimate the size of the rollback segment by running the longest expected transaction and checking the size of the rollback segment. For the current transaction, get the number of blocks used in a rollback segment:

```
SQL> SELECT s.username, t.used_ublk, t.start_time
  2    FROM v$transaction t, v$session s
  3  WHERE t.addr = s.taddr;
```



Sizing Transaction Rollback Data

The number of bytes in rollback segments before execution of statements:

```
SQL> select usn,writes from v$rollstat;
```

After execution of statements:

```
SQL> select usn,writes from v$rollstat;
```

ORACLE

9-16

Sizing Transaction Rollback Data Volume

Another way to estimate the volume of rollback data for a test transaction is to perform the following steps:

1. Before you execute the test transaction, display the current number of writes in the rollback segments:

```
SQL> select usn,writes from v$rollstat;
    USN      WRITES
--------- ---------
        0      1962
        1   1102686
        2     32538
        3   1226096
```

2. Execute the test transaction:

```
SQL> update employees set salary=1000;
6560 rows updated.
```

Sizing Transaction Rollback Data Volume (continued)

3. Display the new number of writes in the rollback segments:

   ```
   SQL> select usn,writes from v$rollstat;
   USN WRITES
   --- -------
   0      1962
   1   2232270
   2     32538
   3   1226096
   ```

   Calculate the difference between the new and the old number of writes in the USN 1 rollback segment to determine the amount of rollback data used for this test transaction.

Using Less Rollback

- **The design of the application should allow users to commit transactions regularly.**
- **Developers should not code long transactions.**

Transactions

You may be able to reduce rollback segment wastage by training users and developers to do the following:

- Users should commit work regularly so that their transactions do not lock others out of rollback segment extents.
- Developers should not code unnecessarily long transactions.

Using Less Rollback

- **Export / Import operations**
  - **Import**
    - **Set `COMMIT = Y`**
    - **Size the set of rows with the `BUFFER` keyword**
  - **Export: Set `CONSISTENT=N`**
- **SQL*Loader operations: Set the commit intervals with `ROWS`**

Developers should make sure that the transactions are not unduly long.

Import

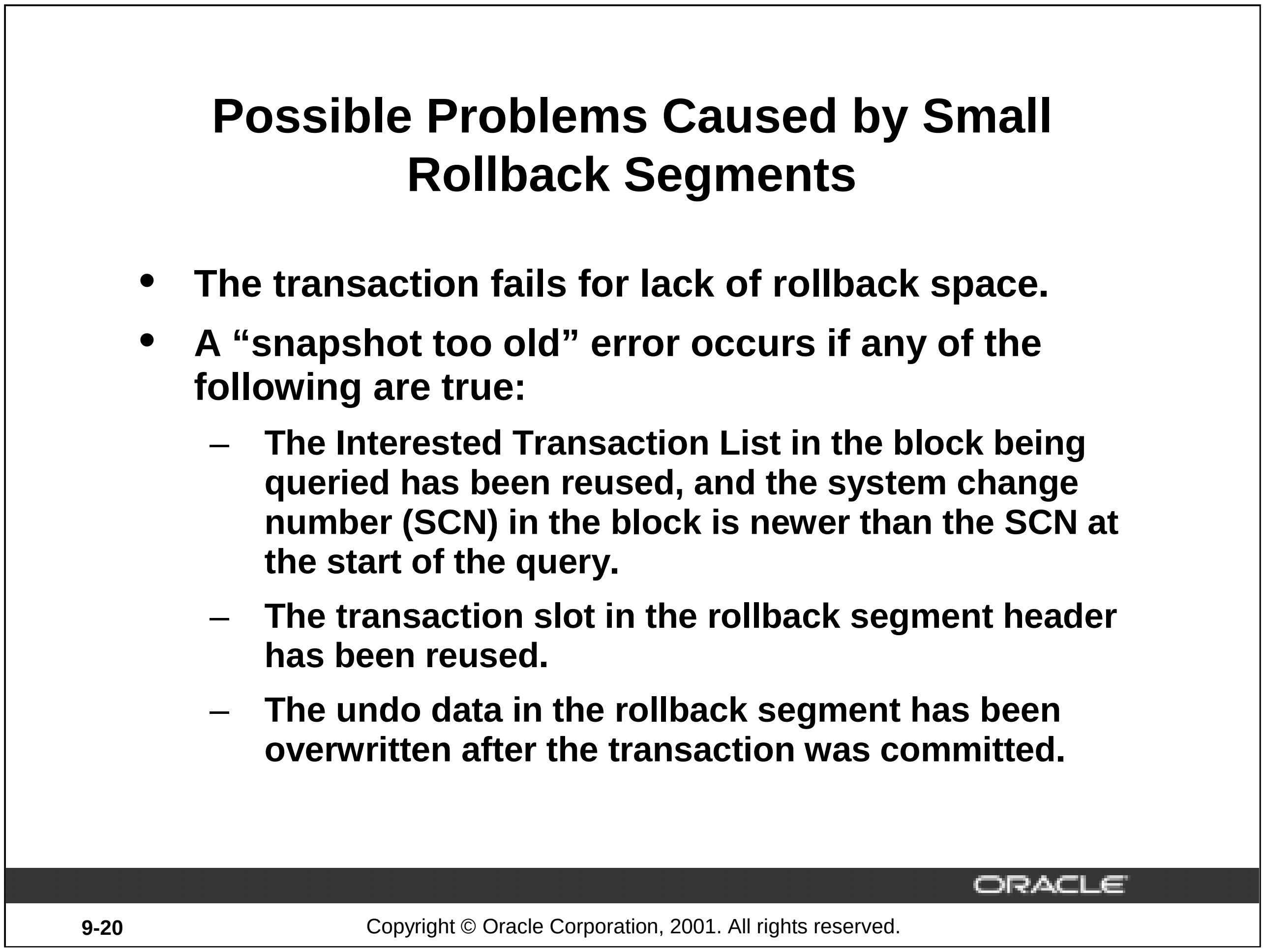
- Set `COMMIT = Y` to make sure that each set of inserted rows is committed as the import goes on.
- Size the set of rows with the `BUFFER_SIZE` keyword.

Export

Setting `CONSISTENT = N` prevents the transaction from being set as read-only, which would use too much rollback segment space. `CONSISTEN=N` specifies whether or not Export uses the `SET TRANSACTION READ ONLY` statement to ensure that the data seen by Export is consistent to a single point in time and does not change during the execution of the export command. You should specify `CONSISTENT=Y` when you anticipate that other applications will be updating the target data after an export has started.

SQL*Loader

For conventional path loading, set the commit intervals with the `ROWS` keyword.



Possible Problems Caused by Small Rollback Segments

- **The transaction fails for lack of rollback space.**
- **A “snapshot too old” error occurs if any of the following are true:**
  – **The Interested Transaction List in the block being queried has been reused, and the system change number (SCN) in the block is newer than the SCN at the start of the query.**
  – **The transaction slot in the rollback segment header has been reused.**
  – **The undo data in the rollback segment has been overwritten after the transaction was committed.**

Large Transactions

The interested transaction list (ITL) resides in the header of the block. Each entry contains the transaction ID that made the change, the undo block address, a flag word, and (depending on the flag), a free space credit and an SCN. The row lock byte contains the ITL entry number which corresponds to the transaction which has the row locked.

If a transaction is exceptionally large, it may fail because the rollback segment cannot expand:

- The rollback segment has reached its maximum number of extents.
- There is no room left in the tablespace for the rollback segment to expand.

You need bigger rollback segments or more space in the tablespace.

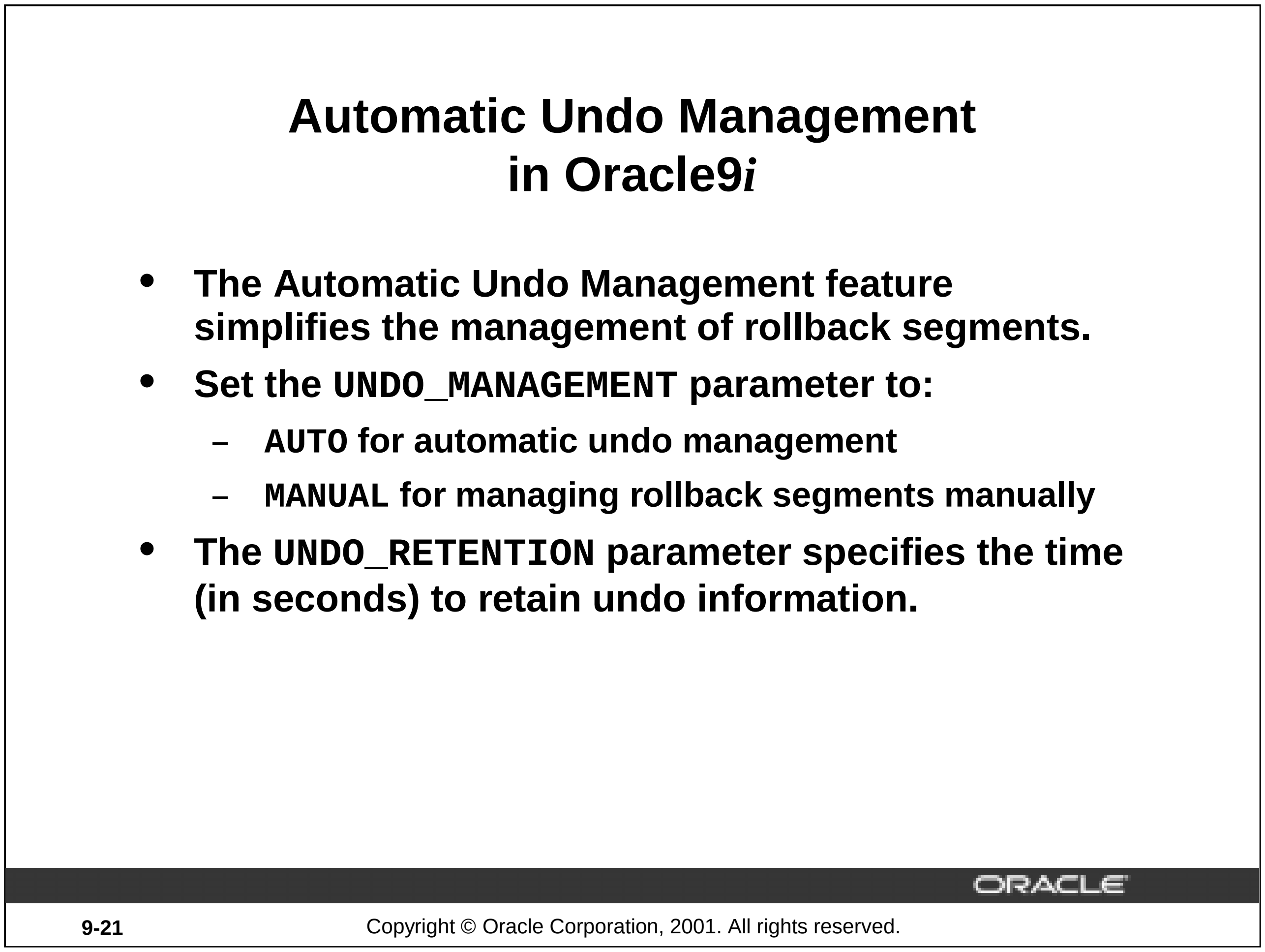
Snapshot Too Old

If a query fails with the following error message, the rollback image needed for read consistency has probably been overwritten by an active transaction:

```
ORA-01555: snapshot too old (rollback segment too small)
```

Occasionally, you may get this error even if there are no other transactions active.

To resolve this error, you need bigger rollback segments. You should also avoid running batch type queries during the day time. If not avoidable, then the long running (queries/load operations) should use a set transaction use rollback segment statement at their beginning.



Automatic Undo Management in Oracle9*i*

- **The Automatic Undo Management feature simplifies the management of rollback segments.**
- **Set the `UNDO_MANAGEMENT` parameter to:**
  - **`AUTO` for automatic undo management**
  - **`MANUAL` for managing rollback segments manually**
- **The `UNDO_RETENTION` parameter specifies the time (in seconds) to retain undo information.**

ORACLE

9-21

Automatic Managed Undo

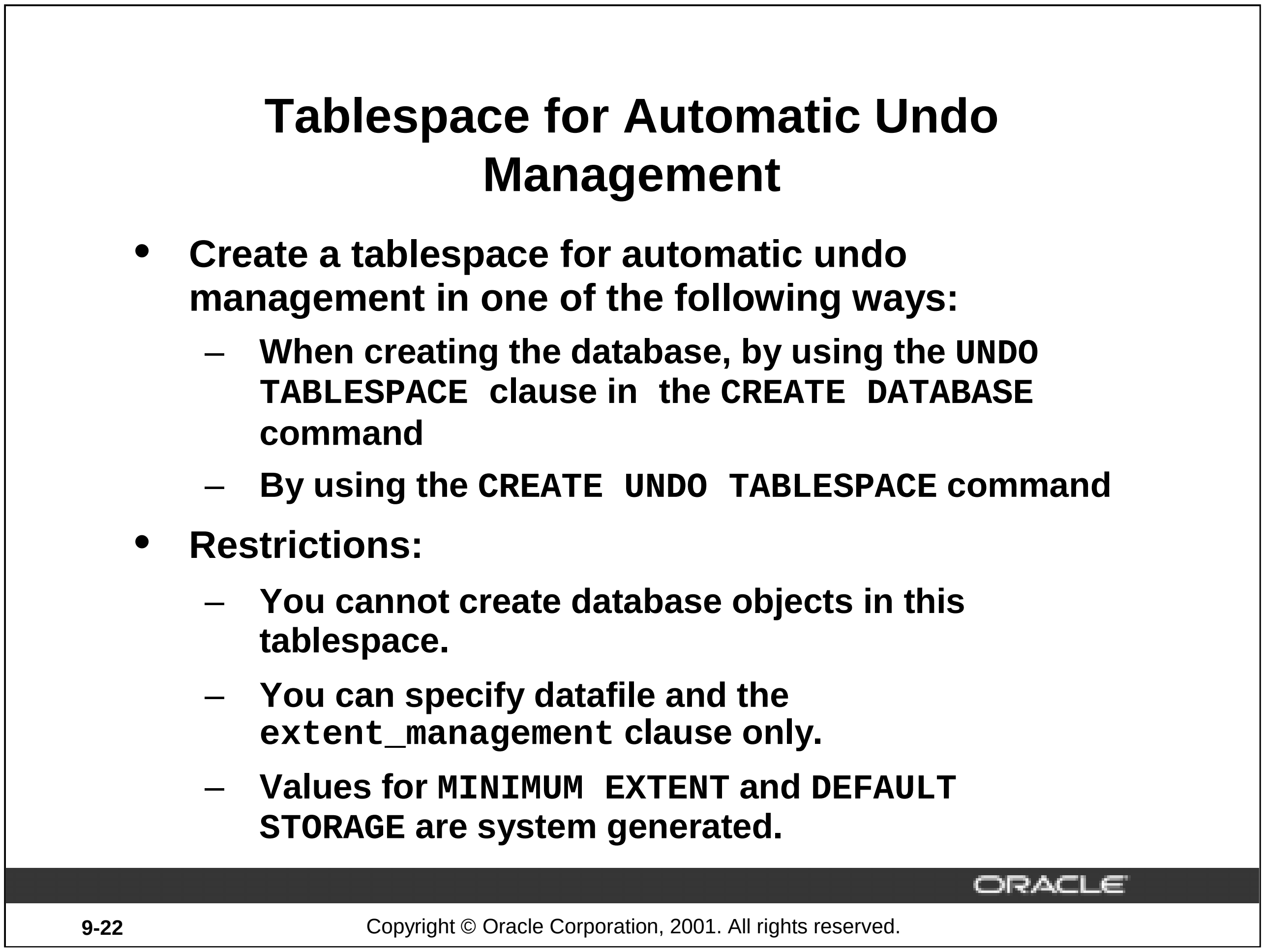
Oracle9*i* provides an automatic undo management feature, which is a new way of managing undo space in Oracle databases. In Oracle8*i* (and earlier versions), undo operations were performed using rollback segments. DBAs managed the number, sizing, and location of rollback segments. This is now called Rollback Segment Undo (RBU).

You have the choice of managing rollback segments automatically or manually. Set the `UNDO_MANAGEMENT` initialization parameter to:

- AUTO to enable the instance to manage rollback segments automatically (AUM)
- MANUAL to create and manage rollback segments manually (RBU)

When the database is set to use auto-managed undo, you need to perform fewer explicit actions for efficient allocation and management of the undo space. Undo segments are created and maintained by Oracle9*i* server. When the first DML operation is executed within a transaction, the transaction is assigned to an undo segment in the current undo tablespace.

You can specify, and accordingly control, the amount of undo information retained in the auto-managed undo segments by using the `UNDO_RETENTION` parameter.



Tablespace for Automatic Undo Management

- **Create a tablespace for automatic undo management in one of the following ways:**
  - **When creating the database, by using the `UNDO TABLESPACE` clause in the `CREATE DATABASE` command**
  - **By using the `CREATE UNDO TABLESPACE` command**
- **Restrictions:**
  - **You cannot create database objects in this tablespace.**
  - **You can specify datafile and the `extent_management` clause only.**
  - **Values for `MINIMUM EXTENT` and `DEFAULT STORAGE` are system generated.**

9-22

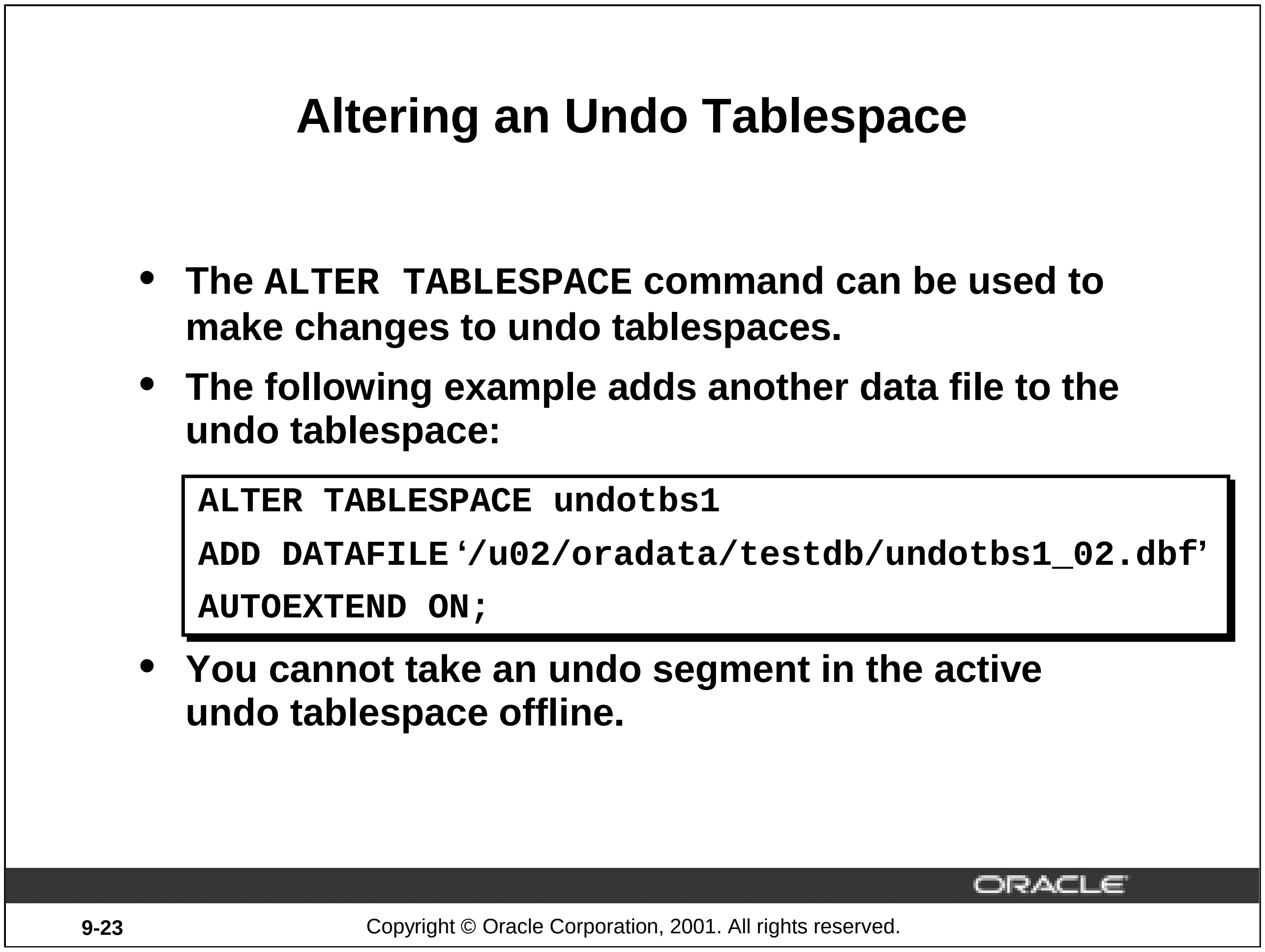
Tablespace for Automatic Undo Management

Set the `UNDO_MANAGEMENT` initialization parameter to AUTO. You can create an auto-managed undo tablespace when creating the database. You can use the `UNDO TABLESPACE` clause to specify the name, data file, size, and block size of the undo tablespace. If you do not specify the `UNDO TABLESPACE` clause when creating the database, then:

- An `UNDO` tablespace with the name `SYS_UNDOTBS` is created.
- The data file with name `DBU1<ORACLE.SID>.dbf` is placed in the `$ORACLE_HOME/dbs` folder.
- `AUTOEXTEND` is set to ON.

You can also create an undo tablespace with the `CREATE UNDO TABLESPACE` command.

You can provide the name of the undo tablespace in `UNDO_TABLESPACE` initialization parameter.



Altering an Undo Tablespace

- **The `ALTER TABLESPACE` command can be used to make changes to undo tablespaces.**
- **The following example adds another data file to the undo tablespace:**

```
ALTER TABLESPACE undotbs1
ADD DATAFILE‘/u02/oradata/testdb/undotbs1_02.dbf’
AUTOEXTEND ON;
```

- **You cannot take an undo segment in the active undo tablespace offline.**

9-23

Altering an Undo Tablespace

The following clauses are supported when altering an undo tablespace:

- `ADD DATAFILE`
- `RENAME`
- `DATAFILE [ONLINE|OFFLINE]`
- `BEGIN BACKUP`
- `ENDBACKUP`

This example depicts the addition of a data file to an existing undo tablespace:

```
ALTER TABLESPACE undotbs1
ADD DATAFILE ‘/u02/oradata/testdb/undotbs1_02.dbf’
AUTOEXTEND ON;
```

Switching Undo Tablespaces

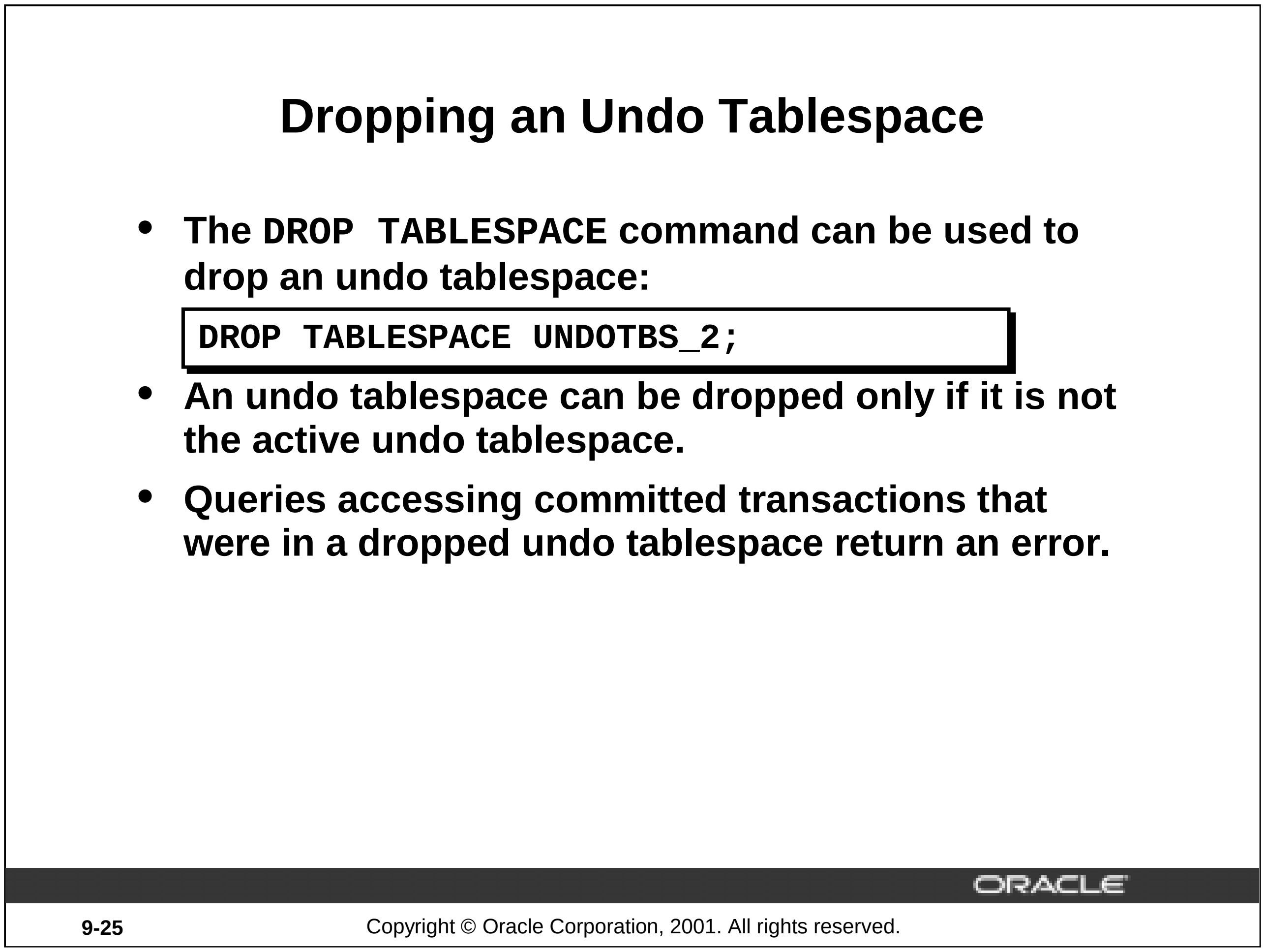
- **A DBA can switch from using one undo tablespace to another.**
- **Only one undo tablespace at a time can be assigned as active.**
- **Switching is performed by using the `ALTER SYSTEM` command:**

```
ALTER SYSTEM SET UNDO_TABLESPACE=UNDOTBS2;
```

Number of Active Undo Tablespaces

At any given moment of time, there can be only one active undo tablespace. However, an instance may have more than one undo tablespace in use on the database. If a database has two undo tablespaces (`UNDOTBS1` and `UNDOTBS2`), only one can be active (in this example: `UNDOTBS1`). This means that all new transactions must use this tablespace to store any undo data.

If the DBA switches the undo tablespace using the `ALTER SYSTEM SET UNDO_TABLESPACE=UNDOTBS2` command, all new transactions are directed to the undo tablespace, `UNDOTBS2`; however, all current transactions (that is, those already assigned to `UNDOTBS1`), will continue to use the undo tablespace `UNDOTBS1`, until they are completed.



Dropping an Undo Tablespace

- The `DROP TABLESPACE` command can be used to drop an undo tablespace:

```
DROP TABLESPACE UNDOTBS_2;
```

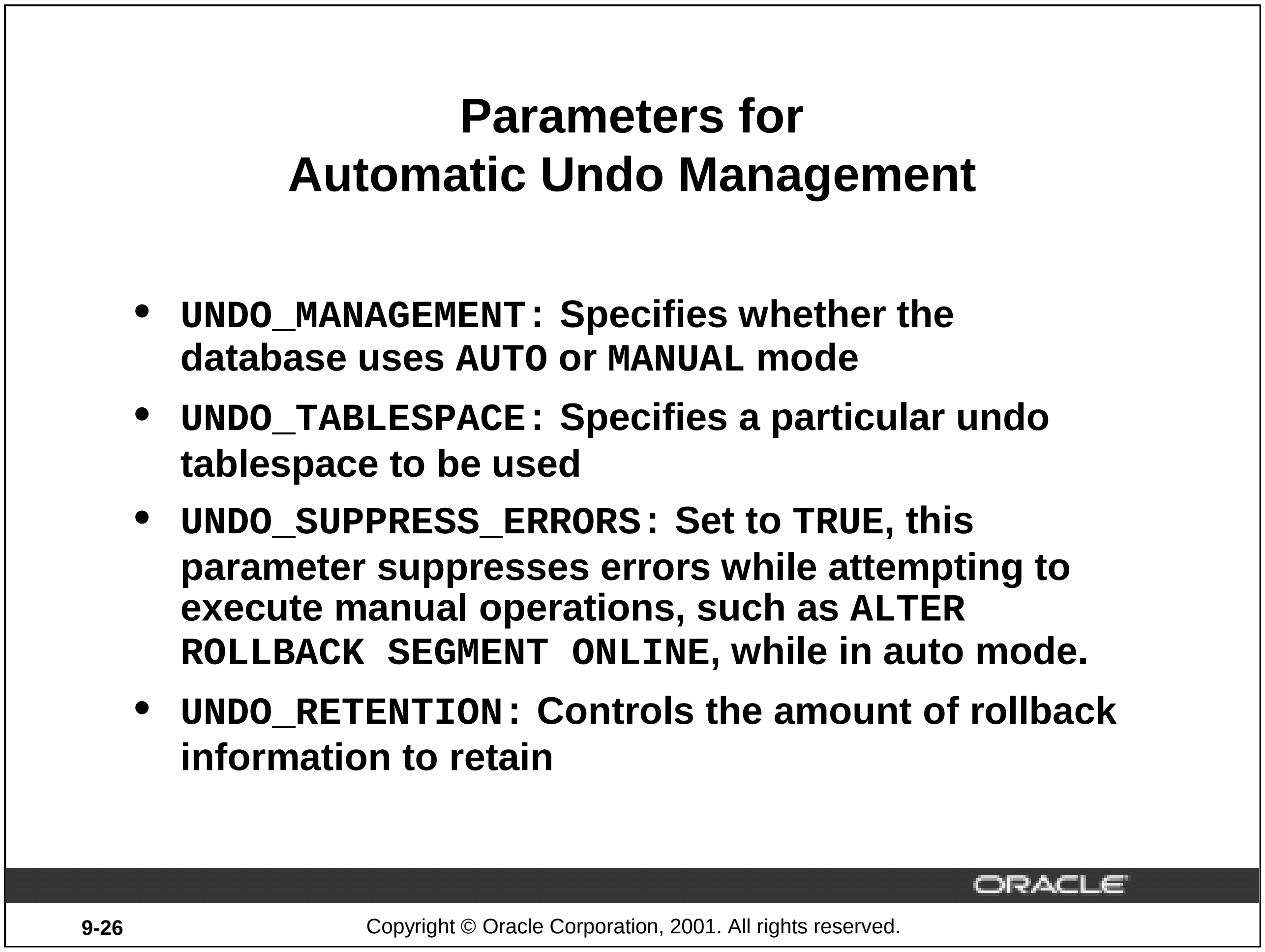
- An undo tablespace can be dropped only if it is not the active undo tablespace.
- Queries accessing committed transactions that were in a dropped undo tablespace return an error.

Dropping an Undo Tablespace

An undo tablespace can be dropped only if:

- it is not currently used by any instance, and
- its transaction tables do not contain any uncommitted transactions.

The `DROP TABLESPACE` *undo tablespace name* command behaves the same as `DROP TABLESPACE` *tablespace name* `INCLUDING CONTENTS.`



Parameters for Automatic Undo Management

- **`UNDO_MANAGEMENT`: Specifies whether the database uses `AUTO` or `MANUAL` mode**
- **`UNDO_TABLESPACE`: Specifies a particular undo tablespace to be used**
- **`UNDO_SUPPRESS_ERRORS`: Set to `TRUE`, this parameter suppresses errors while attempting to execute manual operations, such as `ALTER ROLLBACK SEGMENT ONLINE`, while in auto mode.**
- **`UNDO_RETENTION`: Controls the amount of rollback information to retain**

ORACLE

9-26 Copyright © Oracle Corporation, 2001. All rights reserved.

Parameters for System Managed Undo

The following parameters are used with the System Managed Undo feature:

- `UNDO_MANAGEMENT`: Specifies what mode of undo management to use. The parameter can be reassigned when the database is open.
  - If set to AUTO, the system managed undo feature is used. Make sure that you have already created an undo tablespace.
  - A value of MANUAL means that the rollback segments are managed by the DBA.
- `UNDO_TABLESPACE`: Specifies the name of the undo tablespace
  - If the database is in SMU mode and the `UNDO_TABLESPACE` parameter is omitted at startup, the first available undo tablespace in the database is chosen.
  - If no undo tablespace is available, the instance starts without an undo tablespace using the SYSTEM rollback segment. Make sure that an undo tablespace is available immediately thereafter.
  - To replace one active undo tablespace with another, you can use the `ALTER SYSTEM SET UNDO_TABLESPACE` … command.

Parameters for Automatic Managed Undo (continued)

- `UNDO_SUPPRESS_ERRORS`: Primarily meant for applications and tools that use statements such as `SET TRANSACTION USE ROLLBACK SEGMENT`. Setting this parameter enables users to use the SMU feature before all application programs and scripts are converted to SMU mode. So at the beginning of such sessions, you can add the `ALTER SESSION SET UNDO_SUPPRESS_ERRORS=TRUE` statement to suppress the (OER 30019) error.

- `UNDO_RETENTION`: Controls the amount of committed undo information to retain. You can use `UNDO_RETENTION` to satisfy queries that require old undo information in order to roll back changes to produce older images of data blocks. You can set the value at instance startup.

  The `UNDO_RETENTION` parameter works best if the current undo tablespace has enough space for the active transactions. If an active transaction needs undo space and the undo tablespace does not have any free space, the database starts reusing undo space that would have been retained. This may cause long queries to fail. Be sure to allocate enough space in the undo tablespace to satisfy the space requirement for the current setting of this parameter.

- `UNDO_RETENTION` is specified in units of seconds, with default value of 900 seconds. Because undo segments are on disk, they can survive system crashes. When the instance is recovered, undo information is retained based on the current setting of the `UNDO_RETENTION` initialization parameter. The `UNDO_RETENTION` parameter value can also be changed dynamically using the `ALTER SYSTEM` command:

  ```
  ALTER SYSTEM SET UNDO_RETENTION = 5
  ```

- The effect of the `UNDO_RETENTION` parameter is immediate, but it can be honored only if the current undo tablespace has enough space for the active transactions. If an active transaction requires undo space and the undo tablespace does not have available space, the database starts reusing unexpired undo space. Such action can potentially cause some queries to fail with the *snapshot too old* error.

Space Requirement For Undo Retention

Given a specific `UNDO_RETENTION` parameter setting and some system statistics, the amount of undo space required to satisfy the undo retention requirement can be estimated using the formula:

```
Undo Space = (UNDO_RETENTION * (Undo Blocks Per Second * DB_BLOCK_SIZE) ) +
               DB_BLOCK_SIZE
```

You can use the following query to set the `UNDO_RETENTION` parameter and size the undo tablespace:

```
SELECT (RD * (UPS * OVERHEAD) + OVERHEAD) AS "Bytes“
FROM   (SELECT value AS RD FROM v$parameter
                WHERE  name = 'undo_retention'),
       (SELECT (SUM(undoblks) / SUM( ((end_time - begin_time) * 86400)))
                AS UPS  FROM   v$undostat),
       (SELECT value AS Overhead FROM v$parameter
                WHERE  name = 'db_block_size');
```

Monitoring Automatic Undo Management

- **Use `V$UNDOSTAT` view to monitor undo segments.**
- **This view is available in both manual and auto mode.**
- **The `UndoBlks` column displays the number of undo blocks allocated.**

ORACLE

Monitoring Automatic Managed Undo

Use the `V$UNDOSTAT` view to monitor space allocation and usage for automatically managed undo. Each row in the view keeps statistics collected in the instance for a 10-minute interval.
You can use this view to estimate the amount of undo space required for the current workload. This view is available in both SMU and RBU modes.
The example above shows that the peak undo consumption occurred between 15:30 and 15:40; 1,187 undo blocks were consumed in 10 minutes (or about 2 blocks per second). Also, the highest transaction concurrency occurred during that same period, with 45 transactions executing at the same time. The longest query (1,202 seconds) was executed (and ended) in the period between 15:20 and 15:30.

Two aspects can be tuned under Automatic Undo Management:

- The size of the undo tablespace
- The amount of time that undo blocks are retained before being overwritten.

Summary

In this lesson, you should have learned how to:

- Avoid contention for rollback segment headers
- Work out the appropriate numbers and sizes of rollback segments
- Monitor the rollback space used by transactions
- Monitor an accurate value of the OPTIMAL storage parameter
- Identify possible rollback segment problems
- Use automatic managed undo segments

Practice 9-1

The objective of this practice is to use available diagnostic tools to monitor and tune the rollback segments. This would require setting the database to Manual Undo Management mode.

1. Set the database in Manual Undo Mode.
   a) Connect sys/oracle as SYSDBA and use shutdown immediate to close the database.
   b) Edit the initialization parameter file locate $HOME/ADMIN/PFILE and comment out the following lines
      ```
      undo_management = AUTO
      undo_tablespace = UNDOTBS
      ```
   c) `Save the modifications and startup the database. Confirm that the UNDO_MANAGEMENT is MANUAL, and UNDO_TABLESPACE is null`
2. Connect sys/oracle as SYSDBA and create a new rollback segment tablespace RBS_TEST that is 2 MB in size using the CREATE UNDO TABLESPACE cammand. Name the datafile rbs_test.dbf and place it in the $HOME/ORADATA/u03 directory.
3. For the purposes of this practice, create a new rollback segment called RBSX in the RBS_TEST tablespace. For the storage parameters, use 10 KB for the INITIAL and NEXT extent sizes with MINEXTENTS value set to 20. Set the OPTIMAL value so that the segment shrinks back to 200 KB automatically.
4. Bring the `rbsx` rollback segment online and ensure that any others (except the SYSTEM rollback segment) are offline. Query the DBA_ROLLBACK_SEGS view to get the segment_name and status of the rollback segments to be taken offline using the ALTER ROLLBACK SEGMENT command.
5. Before executing a new transaction, find the number of bytes written so far in the RBSX rollback segment, using the writes column of v$rollstat.
6. Open two sessions. In session 1 connect as hr/hr, and run the script $HOME/STUDENT/LABS/ins_temps.sql. The script inserts 100 new rows into the TEMP_EMPS table – DO NOT COMMIT this transaction. In the second session, log in as system/manager, determine how many rollback segment blocks or bytes the transaction is using? To do this query the writes column of V$ROLLSTAT to get the number of bytes written in the RBSX rollback segment so far. Record this value.
   Note: The number of writes in the rollback segment between questions 5 and 6 is the difference in the value of the writes column at the respective times.
7. Join the V$TRANSACTION and V$SESSION views to find, in the USED_UBLK column, how many blocks the ins_temps transaction is using.
8. Return to the hr session (the first session) and commit the insert. Run the $HOME/STUDENT/LABS/del_temps.sql script – DO NOT COMMIT. The script deletes the hundred rows you have just inserted. As user system (in the second session), check the amount of rollback space used, using the writes column of v$rollstat . Note the difference between the return value, and that found in question 6.

9. Connect as system/manager and find out if you have had any rollback segment contention since startup, using the waits and gets columns in the v$rollstat view.

   Note: In a production environment a better source of information would be ”Rollback Segment” section in the STATSPACK report.

10. Does the V$SYSTEM_EVENT view show any waits related to rollback segments? Query in V$SYSTEM_EVENT view for the “undo segment tx slot” entry.

11. Connect as hr/hr and run the $HOME/STUDENT/LABS/ins_temps.sql script again, allocating the transaction to a specific rollback segment RBSX, using the set transaction use rollback segment command. To check that the transaction is using the defined rollback segment join the V$ROLLSTAT, V$SESSION, and V$TRANSACTION views.

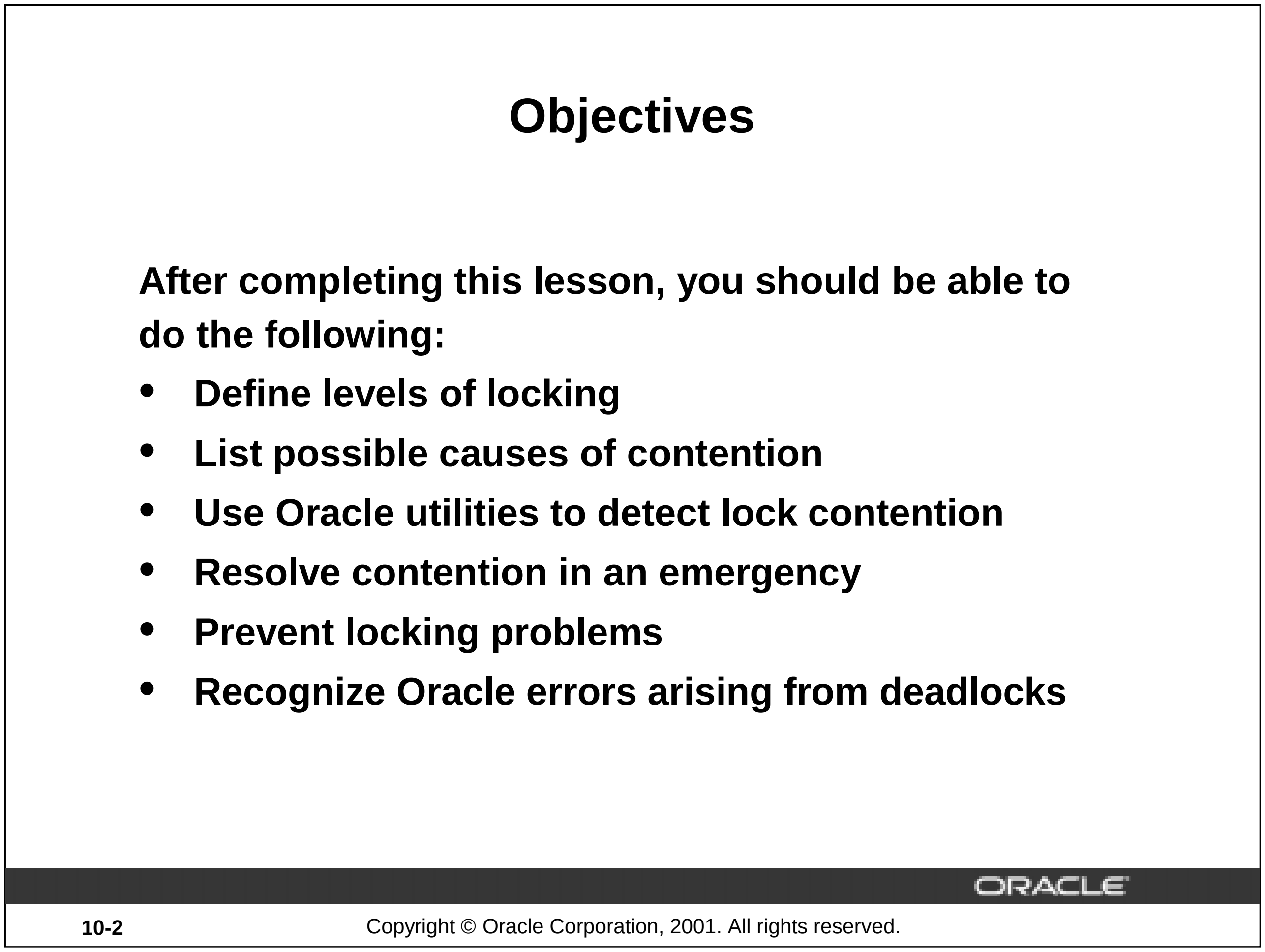
12. Set the database in Auto Undo Mode.

    a) Connect sys/oracle as SYSDBA and use shutdown immediate to close the database.

    b) Edit the initialization parameter file locate $HOME/ADMIN/PFILE and uncomment the following lines
    ```
    undo_management = AUTO
    undo_tablespace = UNDOTBS
    ```

    c) `Save the modifications and startup the database. Confirm that the UNDO_MANAGEMENT is AUTO, and UNDO_TABLESPACE is UNDOTBS`

10 Monitoring and Detecting Lock Contention

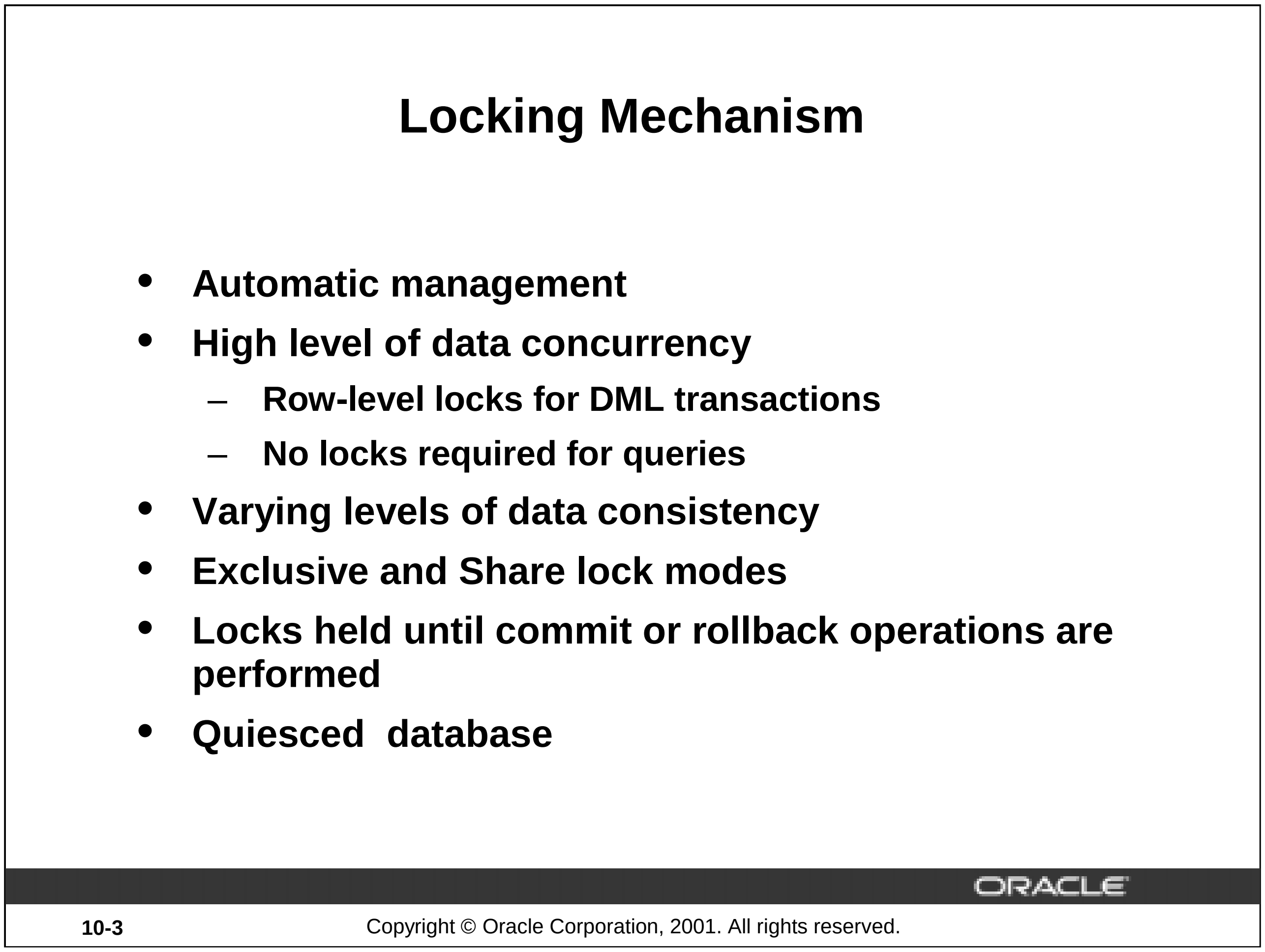


Objectives

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

- Define levels of locking
- List possible causes of contention
- Use Oracle utilities to detect lock contention
- Resolve contention in an emergency
- Prevent locking problems
- Recognize Oracle errors arising from deadlocks

10-2



Locking Mechanism

- Automatic management
- High level of data concurrency
  - Row-level locks for DML transactions
  - No locks required for queries
- Varying levels of data consistency
- Exclusive and Share lock modes
- Locks held until commit or rollback operations are performed
- Quiesced database

10-3

Lock Management

The Oracle server automatically manages locking. The default locking mechanisms lock data at the lowest level of restriction to guarantee data consistency while allowing the highest degree of data concurrency.

Note: The default mechanism can be modified by the `ROW_LOCKING`. The default value is ALWAYS, which leads the Oracle server to always lock at the lowest and least restrictive level (the row level, not the table level) during DML statements. The other possibility is to set the value to INTENT, which leads the Oracle server to lock at a more constraining level (the table level), except for a `SELECT FOR UPDATE` statement, for which a row-level lock is used.

Quiesced Database

Oracle9*i*, Release 1 (9.0.1), enables a DBA to put the system into quiesced state. The system is in quiesced state if there are no active sessions, other than SYS and SYSTEM. An active session is defined as a session that is currently inside a transaction, a query, a fetch, or a PL/SQL procedure, or a session that is currently holding any shared resources, such as enqueues. DBAs are the only users who can proceed when the system is in quiesced state.

Data Concurrency

Locks are designed to allow a high level of *data concurrency*; that is, many users can safely access the same data at the same time.

- Data Manipulation Language (DML) locking is at row level.

Transaction 1	**Transaction 2**

`SQL> UPDATE employee`
`  2  SET salary=salary*1.1`
`  3  WHERE id= 24877;`
`1 row updated.`
	`SQL> UPDATE employee`
	`  2  SET salary=salary*1.1`
	`  3  WHERE id= 24878;`
	`1 row updated.`

- A query holds no locks, unless the user specifies that it should.

Transaction 1	**Transaction 2**

`SQL> UPDATE s_emp`
`  2  SET salary=salary+1200;`
`13120 rows updated.`
	`SQL> SELECT salary`
	` 2   FROM employee`
	` 3   where id = 10;`
	`   SALARY`
	`---------`
	`     1000`

Data Consistency

The Oracle server also provides varying levels of *data consistency*; that is, the user sees a static picture of the data, even if other users are changing it.

Duration

Locks are held until the transaction is committed, rolled back, or terminated. If a transaction terminates abnormally, the `PMON` process cleans up the locks.

Locking Modes

- Exclusive lock mode prevents the associated resource from being shared with other transactions, until the exclusive lock is released.

  Example: Exclusive locks are set at row level for a DML transaction:

Transaction 1	**Transaction 2**
`SQL> UPDATE employee`	`SQL> UPDATE employee`
`  2  SET salary=salary*1.1`	`  2  SET salary=salary*1.1`
`  3  WHERE id= 24877;`	`  3  WHERE id= 24877;`
`1 row updated.`	`Transaction 2 waits.`

- In Share lock mode, several transactions can acquire share locks on the same resource.

  Example: Shared locks are set at table level for DML transactions:

Transaction 1	**Transaction 2**
`SQL> UPDATE employee`	`SQL> UPDATE employee`
`  2  SET salary=salary*1.1`	`  2  SET salary=salary*1.1`
`  3  WHERE id= 24877;`	`  3  WHERE id= 24878;`
`1 row updated.`	`1 row updated.`

The two transactions update rows in the same table.

Lock Duration

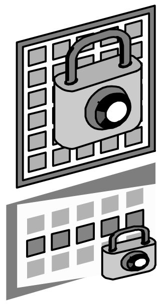
- Transactions hold locks until the transactions are committed or rolled back:

Transaction 1	**Transaction 2**
`SQL> UPDATE employee`	`SQL> UPDATE employee`
`  2  SET salary=salary*1.1`	`  2  SET salary=salary*1.1`
`  3  WHERE id= 24877;`	`  3  WHERE id= 24877;`
`1 row updated.`	**`Transaction 2 waits until`**
`SQL> commit;`	**`transaction 1 is committed.`**
`Commit complete.`	`1 row updated.`

As soon as Transaction 1 is committed, Transaction 2 can update the row, because the transaction acquired the requested lock.

Two Types of Locks

- DML or data locks:
  - Table-level locks → (TM)
  - Row-level locks → (TX)



- DDL or dictionary locks

DML Locks

DML locks guarantee the integrity of data being accessed concurrently by multiple users for incorporating changes. They prevent destructive interference of simultaneous conflicting DML and DDL operations.

DML Levels: A *table-level* lock (TM type) is set for any DML transaction that modifies a table: `INSERT`, `UPDATE`, `DELETE`, `SELECT...FOR UPDATE`, or `LOCK TABLE`. The table lock prevents DDL operations that would conflict with the transaction.

Example

Transaction 1	Transaction 2

`SQL> UPDATE employee`
`  2   SET salary=salary*1.1;`
`13120 rows updated.`
	`SQL> DROP TABLE employee;`
	`ERROR at line 1:`
	`ORA-00054: resource busy and acquire with NOWAIT specified`

DML Locks (continued)

The *row-level* lock (TX type) is automatically acquired for each row modified by `INSERT`, `UPDATE`, `DELETE`, or `SELECT...FOR UPDATE` statements. The row-level lock ensures that no other user can modify the same row at the same time. Therefore, there is no risk that a user can modify a row that is being modified and not yet committed by another user.

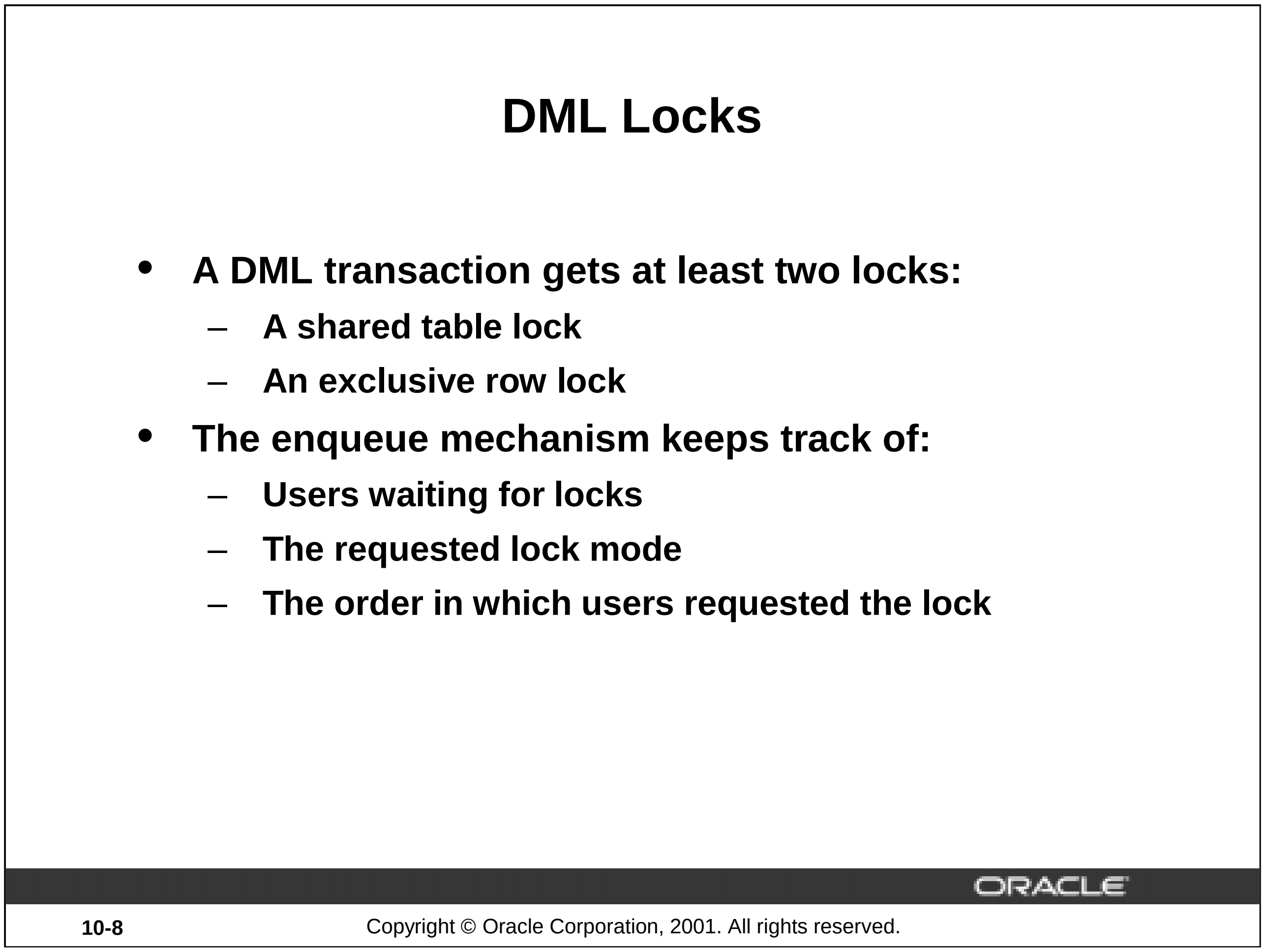
Example

Transaction 1	**Transaction 2**

`SQL> update employee`
`  2   set salary=salary*1.1`
`  3   where id= 24877;`
`1 row updated.`
	`SQL> update employee`
	`  2   set salary=salary*1.1`
	`  3   where id= 24877;`
	`Transaction 2 waits.`

DDL Locks

A DDL lock protects the definition of a schema object while that object is acted upon or referred to by an ongoing DDL operation. The Oracle server automatically acquires a DDL lock to prevent any destructive interference from other DDL operations that might modify or reference the same schema object.



DML Locks

- **A DML transaction gets at least two locks:**
  - **A shared table lock**
  - **An exclusive row lock**
- **The enqueue mechanism keeps track of:**
  - **Users waiting for locks**
  - **The requested lock mode**
  - **The order in which users requested the lock**

DML Transactions Acquire at Least Two Locks

Two kinds of lock structures are used for DML statements (`INSERT`, `UPDATE`, `DELETE`, or `SELECT...FOR UPDATE`):

- The transaction gets a shared lock on the table that is referenced as a TM lock, no matter what shared lock mode it is.
- The transaction gets an exclusive lock on the rows it is changing, referenced as a TX lock. Each row gets a lock byte turned on in the row header pointing to the ITL slot used by the transaction. The lock mode at row level can only be exclusive.

Enqueue Mechanism

The Oracle server maintains all locks as *enqueues*. The enqueue mechanism can keep track of:

- Users waiting for locks held by other users
- The lock mode these users require
- The order in which users requested the lock

If three users want to update the same row at the same time, all of them get the shared table lock, but only one (the first) gets the row lock. The table-locking mechanism keeps track of who holds the row lock and who waits for it.

You can increase the overall number of locks available for an instance by increasing the values of the `DML_LOCKS` and `ENQUEUE_RESOURCES` parameters. This may be necessary in a parallel server configuration.

Table Lock Modes

These table lock modes are automatically assigned by the Oracle server:

- **Row Exclusive (RX): `INSERT, UPDATE, DELETE`**
- **Row Share (RS): `SELECT... FOR UPDATE`**

Automatic Table Lock Modes

You often see the two TM table lock modes held by DML transactions, RX and RS.

These are the table lock modes automatically assigned by the Oracle server for DML transactions.

The restrictiveness of a table lock's mode determines the modes in which other table locks on the same table can be obtained and held.

Row Exclusive (RX)

- Permits other transactions to query, insert, update, delete, or lock other rows concurrently in the same table
- Prevents other transactions from manually locking the table for exclusive reading or writing

Example

Transaction 1 (RX table lock held)	Transaction 2 (RX table lock held)

`SQL> update employee`
`  2   set salary=salary*1.1`
`  3   where id= 24877;`
`1 row updated.`
	`SQL> update employee`
	`  2   set salary=salary*1.1`
	`  3   where id= 24878;`
	`1 row updated.`

Automatic Table Lock Modes (continued)

Row Share (RS)

You can choose to lock rows during a query by using the `SELECT ... FOR UPDATE` statement.

This prevents other transactions from manually locking the table for exclusive write access.

Example

Transaction 1 (RS table lock held)	**Transaction 2 (RX table lock held)**

`SQL> select id,salary`
`  2  from employee`
`  3  where id=24877`
`  4  for update;`
`       ID    SALARY`
`--------- ---------`
`    24877      1100`
`SQL> commit;`
`Commit complete.`
	`SQL> lock table employee`
	`  2   in exclusive mode;`
	`Transaction 2 waits.`
	
	`Table(s) Locked.`

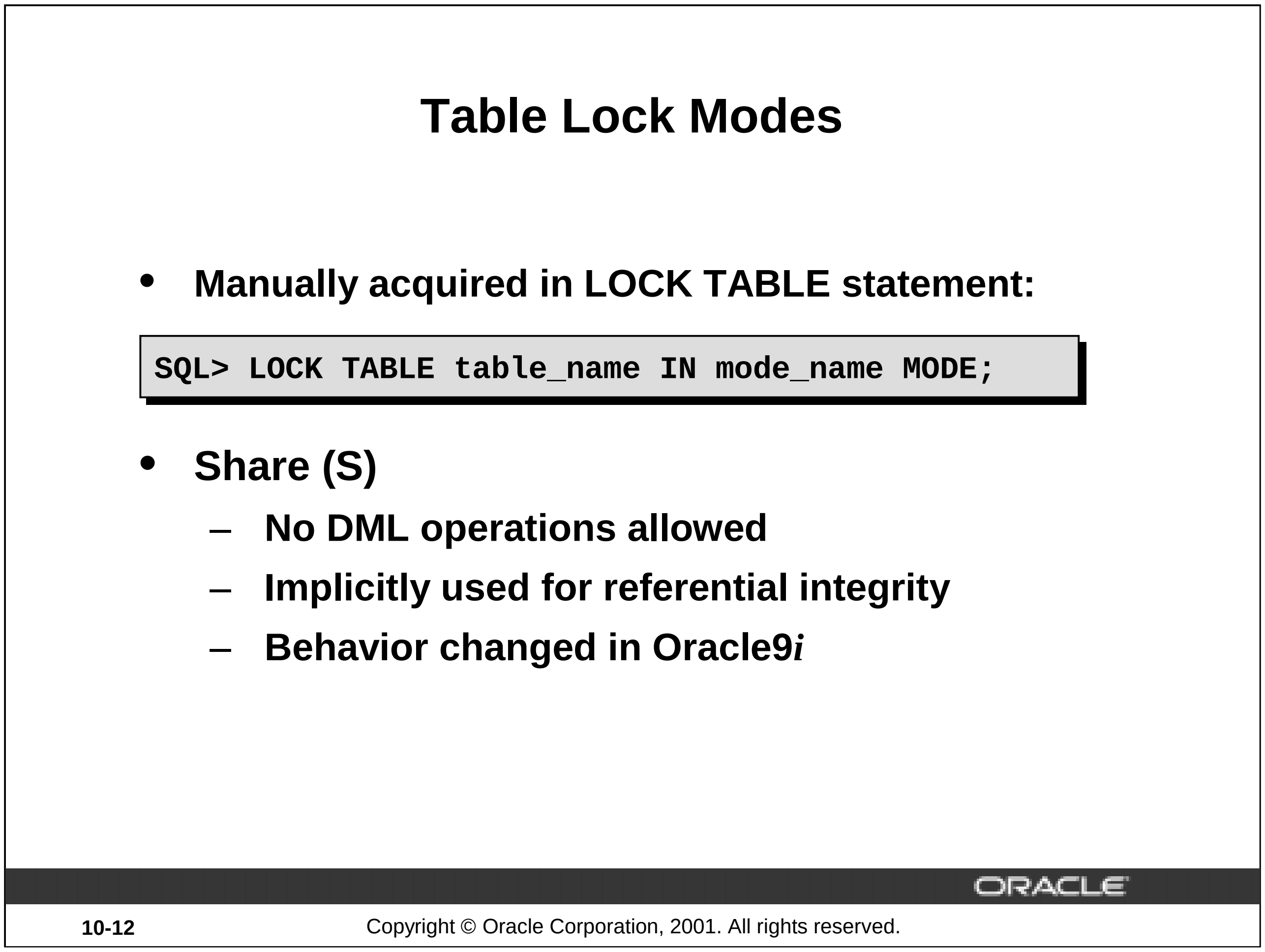


Table Lock Modes

- **Manually acquired in LOCK TABLE statement:**

```
SQL> LOCK TABLE table_name IN mode_name MODE;
```

- **Share (S)**
  - **No DML operations allowed**
  - **Implicitly used for referential integrity**
  - **Behavior changed in Oracle9*i***

10-12 Copyright © Oracle Corporation, 2001. All rights reserved.

Manual Table Lock Modes

The three other table lock modes are assigned manually by an explicit `LOCK TABLE` command. For example:

```
SQL> LOCK TABLE employee IN exclusive MODE;
Table(s) Locked.
```

Often there are good application reasons for explicit locking, but if you get lock contention you may want to check with the developers.

Non-Oracle developers sometimes use unnecessarily high locking levels.

Share (S) Lock Mode

This lock mode permits other transactions to only query the `SELECT ... FOR UPDATE` table. It prevents any modification to the table.

The SQL statements that implicitly get a share lock involve referential integrity constraints. In Oracle9*i*, the implementation of the referential integrity constraint has changed. It does not require the index on foreign key column of the child table.

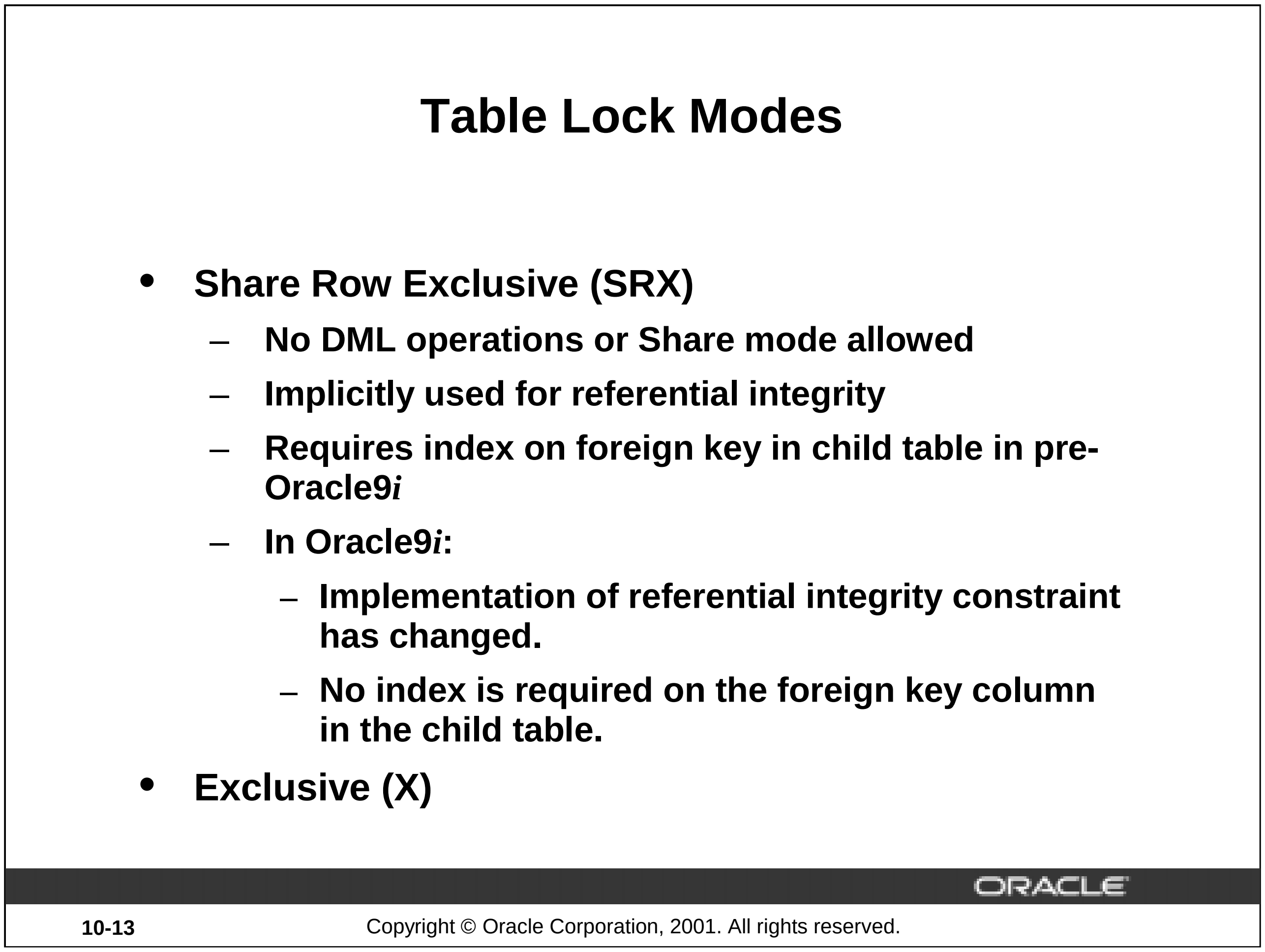


Table Lock Modes

- **Share Row Exclusive (SRX)**
  - **No DML operations or Share mode allowed**
  - **Implicitly used for referential integrity**
  - **Requires index on foreign key in child table in pre-Oracle9*i***
  - **In Oracle9*i*:**
    - **Implementation of referential integrity constraint has changed.**
    - **No index is required on the foreign key column in the child table.**
- **Exclusive (X)**

Share Row Exclusive (SRX) Lock Mode

This is an even higher level of table lock, which prevents DML statements and the manual share lock mode from being acquired.The SQL statement that implicitly gets a Share Row Exclusive lock again involves referential integrity.

Note: In pre-Oracle9*i* releases, the solution is to index the foreign key column on the child table. In Oracle9*i*, the implementation of foreign key constraint is modified and you need not create an index on the foreign key column on the child table.

Exclusive (X) Lock Mode

This is the highest level of table lock, thus the most restrictive mode. Exclusive table lock:

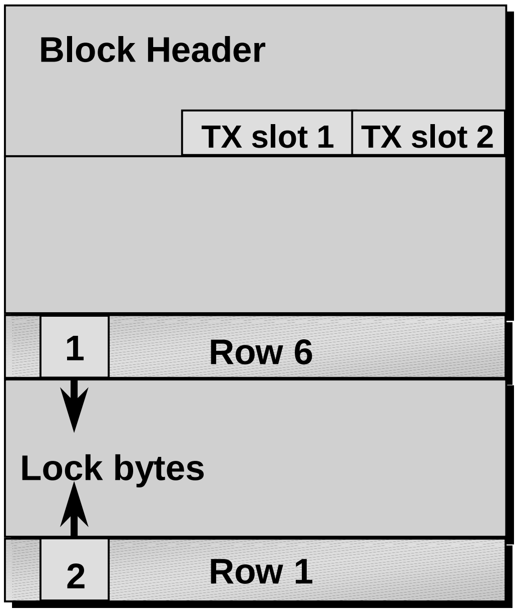
- Permits other transactions only to query the table
- Prevents any type of DML statements and any manual lock mode

Example

Transaction 1 (X table lock held)	**Transaction 2 (RX table lock requested)**

`SQL> LOCK TABLE department IN`
`EXCLUSIVE MODE;`
`Table(s) Locked.`
	`SQL> SELECT * from department`
	`  2   FOR UPDATE;`
	`Transaction 2 waits.`

DML Locks in Blocks



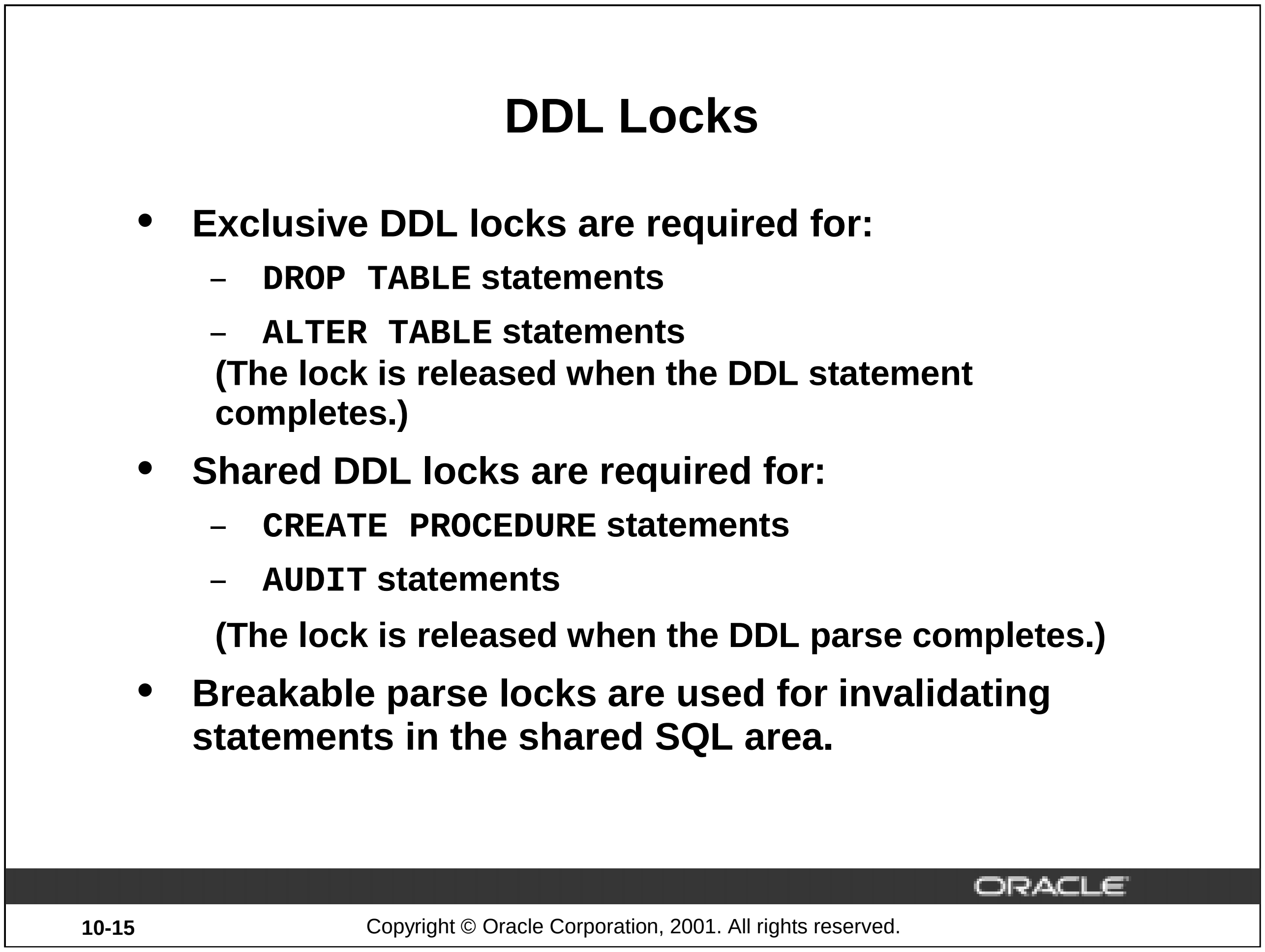
Technical Note

This locking information is not cleared out when transactions are committed, but rather when the next query reads the block. This is known as delayed block cleanout.

The query that does the cleaning must check the status of the transaction and the system change number (SCN) in the transaction table held in the rollback segment header.

Within blocks, the Oracle server keeps an identifier for each active transaction in the block header. At row level, the lock byte stores an identifier for the slot containing the transaction.

Example: In the diagram shown on the slide, the transaction using slot 1 is locking row 6, and the transaction in slot 2 is locking row 1.



DDL Locks

- **Exclusive DDL locks are required for:**
  - **`DROP TABLE` statements**
  - **`ALTER TABLE` statements**

  (The lock is released when the DDL statement completes.)
- **Shared DDL locks are required for:**
  - **`CREATE PROCEDURE` statements**
  - **`AUDIT` statements**

  (The lock is released when the DDL parse completes.)
- **Breakable parse locks are used for invalidating statements in the shared SQL area.**

DDL Locks

You are unlikely to see contention for DDL locks because they are held only briefly and are requested in NOWAIT mode. There are three types of DDL locks.

Exclusive DDL Locks

Some DDL statements, such as CREATE, `ALTER`, and `DROP`, must get an exclusive lock on the object they are working on.

Users cannot get an exclusive lock on the table if any other user holds any level of lock, so an `ALTER TABLE` statement fails if there are users with uncommitted transactions on that table.

Example

Transaction 1	**Transaction 2**

`SQL> UPDATE employee`
`  2  SET salary=salary*1.1;`
`3120 rows updated.`
	`SQL> ALTER TABLE employee`
	`  2  DISABLE PRIMARY KEY;`
	`ORA-00054: resource busy and`
	`acquire with NOWAIT specified`

DDL Locks (continued)

Exclusive DDL Locks (continued)

Note: Using locally managed tablespaces instead of dictionary-managed tablespaces eliminates the contention for the space transaction lock because locally managed tablespaces do not update tables in the data dictionary when asking for space allocation.

Shared DDL Locks

Some statements, such as `GRANT` and `CREATE PACKAGE`, need a shared DDL lock on the objects they reference.

This type of lock does not prevent similar DDL statements, or any DML statements, but it prevents another user from altering or dropping the referenced object.

Breakable Parse Locks

A statement or PL/SQL object in the library cache holds one of these locks for every object it references, until the statement is aged out of the shared pool.

The *breakable parse lock* is there to check whether the statement should be invalidated if the object changes.

You could think of this lock as a pointer. It never causes waits or contention.

Possible Causes of Lock Contention

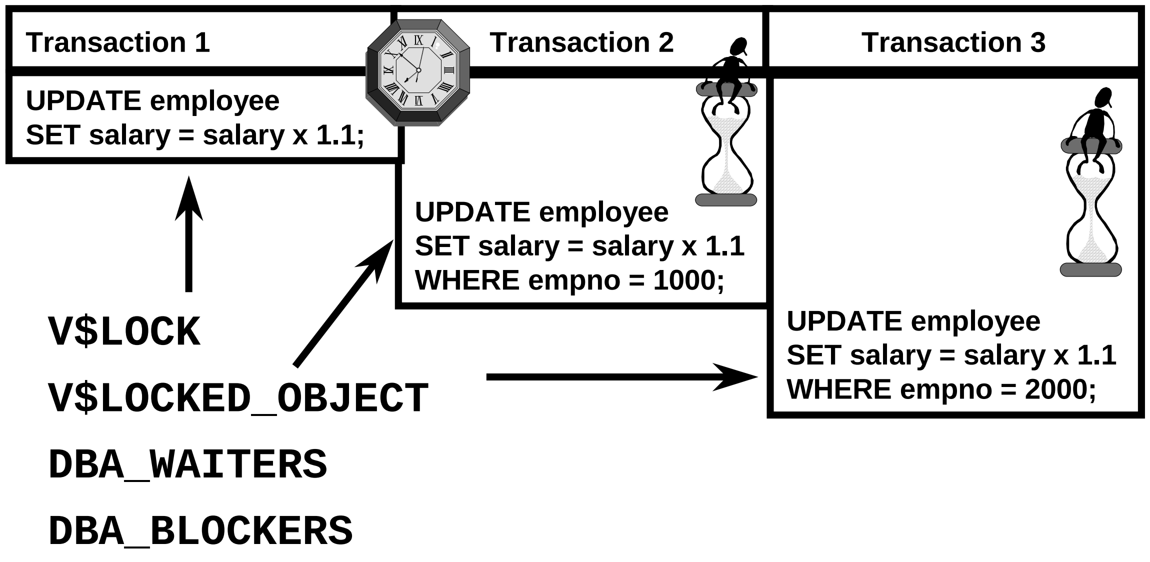
- **Unnecessarily high locking levels**
- **Long-running transactions**
- **Uncommitted changes**
- **Other products imposing higher-level locks**

Development and User Issues

The Oracle server locks are inexpensive and efficient, and most sites do not have problems with locking. If locks do cause contention, it is often because:

- Developers have coded in unnecessarily high locking levels
- Developers have coded in unnecessarily long transactions
- Users are not committing changes when they should
- The application uses the Oracle server in conjunction with other products that impose higher locking levels

Diagnostics Tools for Monitoring Locking Activity



`DBA_WAITERS` and `DBA_BLOCKERS`

These views give further insight into who is holding or waiting for which tables. In order to create these views the script `CATBLOCK.SQL` needs to be run. It is found in the `$ORACLE_HOME/rdbms/admin` directory.

The `V$LOCK` View

Lock type	*ID1*
TX	**Rollback Segment number and slot number**
TM	**Object ID of the table being modified**

Example

To find the table name that corresponds to a particular resource ID 1 of the `V$LOCK` view:

```
SQL> SELECT owner, object_id, object_name, object_type,
  2         v$lock.type
  3  FROM dba_objects, v$lock
  4  WHERE object_id = v$lock.id1 and object_name = table_name;
```

Any process that is blocking others is likely to be holding a lock obtained by a user application. The locks acquired by user applications are:

- Table locks (TM)
- Row-level locks (TX)

Other locks not listed here are system locks that are held only briefly.

The V$LOCKED_OBJECT View

Lock Type	ID1
XIDUSN	Rollback segment number
OBJECT_ID	ID of the object being modified
SESSION_ID	ID of the session locking the object
ORACLE_USERNAME	
LOCKED_MODE	

Example

To find the table name that corresponds to a particular object ID in the V$LOCKED_OBJECT view:

```
SQL> SELECT xidusn, object_id, session_id, locked_mode
  2  FROM v$locked_object;

   XIDUSN OBJECT_ID SESSION_ID LOCKED_MODE
--------- --------- ---------- -----------
        3      2711          9           3
        0      2711          7           3


SQL> SELECT object_name FROM dba_objects
  2  WHERE object_id = 2711;

OBJECT_NAME
-------------
EMPLOYEE
```

If the value of XIDUSN is 0, then the session with the corresponding session ID is requesting and waiting for the lock being held by the session, for which XIDUSN value is different from 0.

UTLLOCKT Script

You can also use the UTLLOCKT.SQL script located in $ORACLE_HOME/rdbms/admin to display lock wait-for in a hierarchy. The script prints the sessions that are waiting for locks and the sessions that are blocking.

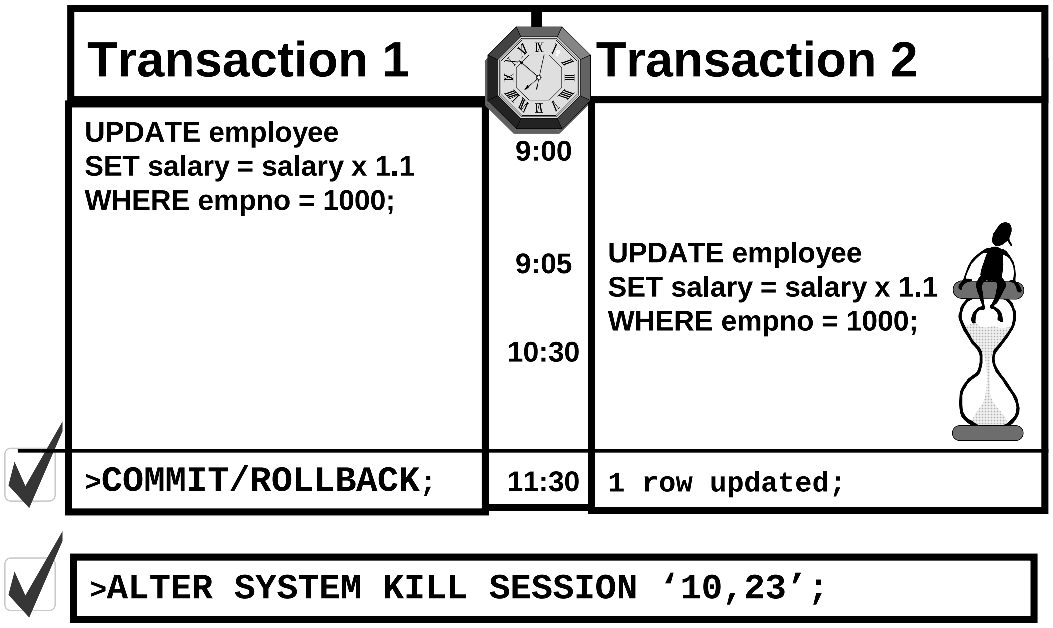
You must have run the CATBLOCK.SQL script (found in $ORACLE_HOME/rdbms/admin folder) as sysdba user before using UTLLOCKT.SQL. CATBLOCK.SQL creates the views, dba_locks and dba_blockers, along with others that will be used by UTLLOCKT.SQL.

For example, in the following output session 9 is waiting for session 8, Sessions 7 and 10 are waiting for 9.

```
WAITING_SESSION   TYPE MODE REQUESTED    MODE HELD       LOCK ID1 LOCK ID2
----------------- ---- ----------------- --------------- -------- --------
8                 NONE None              None            0        0
    9             TX   Share (S)         Exclusive (X)   65547    16
       7          RW   Exclusive (X)     S/Row-X (SSX)   33554440  2
      10          RW   Exclusive (X)     S/Row-X (SSX)   33554440  2
```

Note: The lock information to the right of the session ID describes the lock that the session is waiting for (not the lock it is holding).

Guidelines for Resolving Contention



Killing Sessions

If a user is holding a lock required by another user, you can:

- Contact the holder and ask this user to commit or roll back the transaction
- As a last resort, kill the Oracle user session; this rolls back the transaction and releases locks

Any of the monitoring methods detailed above will give you the session identifier for the user.

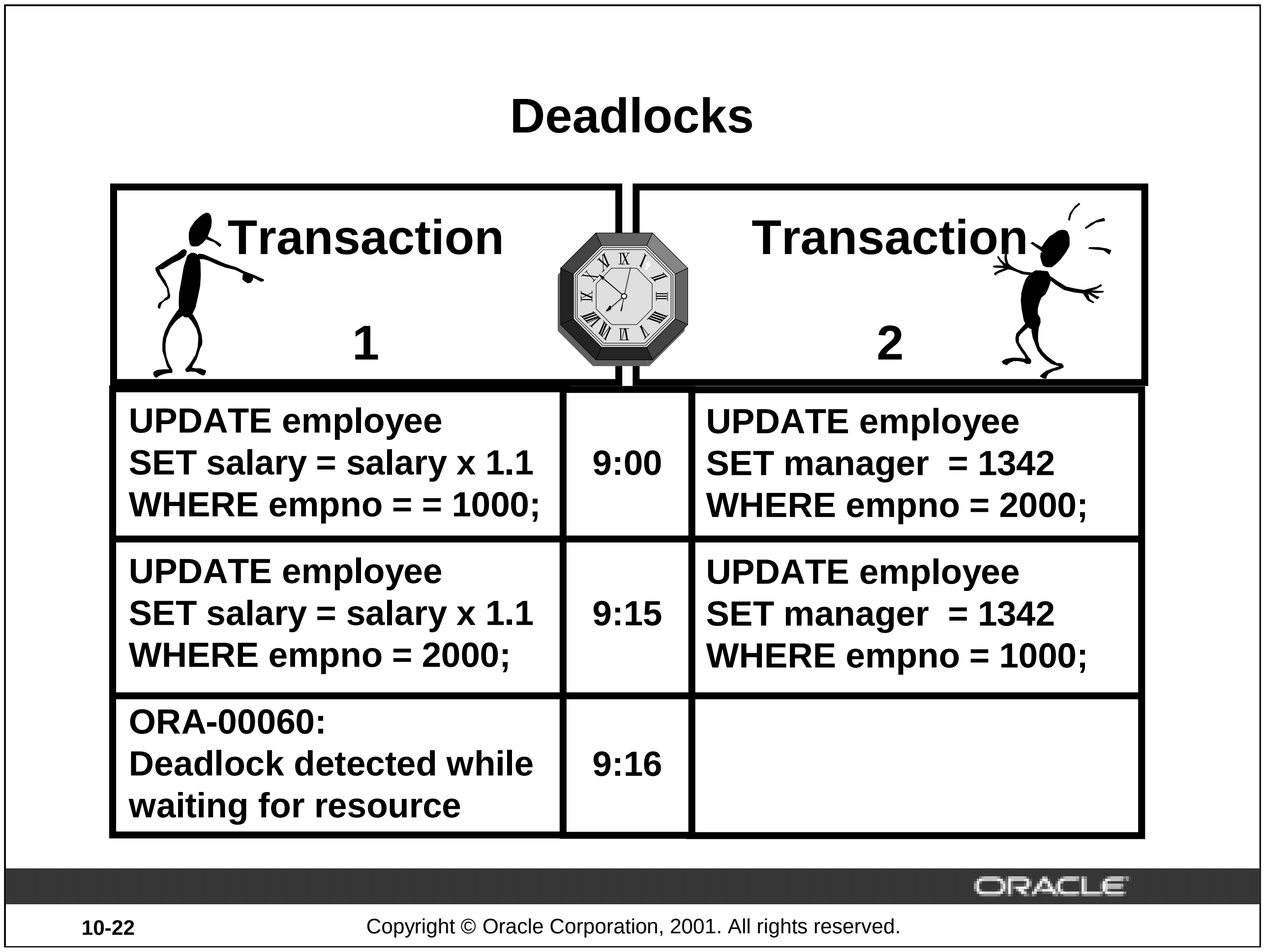
You can kill user sessions with the `ALTER SYSTEM KILL SESSION SQL` command:

```
SQL> SELECT sid,serial#,username
  2  FROM v$session
  3  WHERE type=’USER’;
      SID   SERIAL#   USERNAME
--------- --------- ------------------------------
        8       122   SYSTEM
       10        23   SCOTT
SQL> ALTER SYSTEM KILL SESSION '10,23';
System altered.
```

Which Row Is Causing Contention?

If you need to know which row is causing contention, the `V$SESSION` view contains the following columns:

- `row_wait_block#`
- `row_wait_row#`
- `row_wait_file#`
- `row_wait_obj#`



Deadlocks

Transaction 1		Transaction 2

UPDATE employee
SET salary = salary x 1.1
WHERE empno = = 1000;
	9:00	UPDATE employee
		SET manager = 1342
		WHERE empno = 2000;

UPDATE employee
SET salary = salary x 1.1
WHERE empno = 2000;
	9:15	UPDATE employee
		SET manager = 1342
		WHERE empno = 1000;

ORA-00060: Deadlock detected while waiting for resource	9:16	

10-22

Deadlocks

A *deadlock* can arise when two or more users wait for data locked by each other.

The Oracle server automatically detects and resolves deadlocks by rolling back the statement that detected the deadlock.

Transaction 1	Time	Transaction 2

`SQL> UPDATE employee`
`  2  SET salary=salary*1.1`
`  3  WHERE id= 24877;`
`1 row updated.`
	1	`SQL> UPDATE employee`
		`  2  SET salary=salary*1.1`
		`  3  WHERE id= 24876;`
		`1 row updated.`

`SQL> UPDATE employee`
`  2  SET salary=salary*1.1`
`  3  WHERE id= 24876;`
Transaction 1 waits.
	2	`SQL> UPDATE employee`
		`  2  SET salary=salary*1.1`
		`  3  WHERE id= 24877;`
		Transaction 2 waits.

`ORA-00060: deadlock detected while waiting for resource`	3	

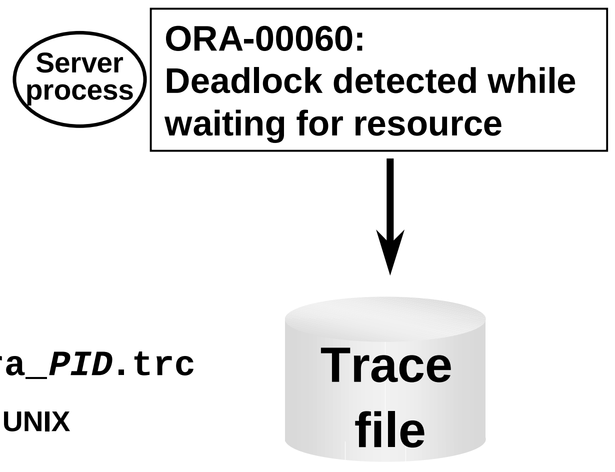
Deadlocks (continued)

If the second update in Transaction 1 detects the deadlock, the Oracle server rolls back that statement and returns the message. Although the statement that caused the deadlock is rolled back, the transaction is not, and you receive an ORA-00060 error. Your next action should be to roll back the remainder of the transaction.

Technical Note

Deadlocks most often occur when transactions explicitly override the default locking of the Oracle server. Distributed deadlocks are handled in the same way as nondistributed deadlocks.

Deadlocks



***SID*_ora_*PID*.trc**

UNIX

in `USER_DUMP_DEST` directory

Trace File

A deadlock situation is recorded in a trace file in the `USER_DUMP_DEST` directory. It is advisable to monitor trace files for deadlock errors to determine whether there are problems with the application. The trace file contains the row IDs of the locking rows.

In distributed transactions, local deadlocks are detected by analyzing a “waits for” graph, and global deadlocks are detected by a time-out.

Once detected, nondistributed and distributed deadlocks are handled by the database and application in the same way.

Summary

In this lesson, you should have learned that:

- Queries do not lock data, except by choice.
- DML statements use row-level and table-level locks on tables.
- Exclusive locks are rarely used.
- You can monitor locks using the `V$LOCK`, `V$LOCKED_OBJECT`, `DBA_WAITERS`, and `DBA_BLOCKERS` views.

Practice 10

The objective of this practice is to use available diagnostic tools to monitor lock contention.

You will need to start three sessions, in separate windows. Log in as hr/hr in two separate sessions (sessions 1 and 2) and as sys/oracle as sysdba in a third session (session 3).

1. In session 1 (user hr/hr), update the salary by 10% for all employee with a salary < 15000 in the temp_emps table. DO NOT COMMIT.
2. In session 3 (sys/oracle as sysdba) check to see if any locks are being held by querying the V$LOCK.
3. In session 2 (user hr/hr – the session not yet used) drop the TEMP_EMPS table. Does it work?

Note: The DDL statement requires an exclusive table lock. It cannot obtain it, because session 1 already holds a row exclusive table lock on the TEMP_EMPS table.

4. In session 2 (hr/hr), update the salary by 5% for all employee with a salary > 15000 in the temp_emps table. DO NOT COMMIT.
5. In session 3, check to see what kind of locks are being held on the TEMP_EMPS table, using the V$LOCK view.
6. Roll back the changes you made in the second session (using hr/hr), and set the manager_id column to 10 for all employees who have a salary < 15000.
7. In session 3, check to see what kind of locks are being held on the TEMP_EMPS table, using the V$LOCK view.
8. In session 3, connect run the script $ORACLE_HOME/rdbms/admin/catblock.sql. The script will create a view DBA_WAITERS, that gives information regarding sessions holding or waiting on a lock. Use this view to determine the session id for the session that is holding locks. Use this value to query v$session to get the serial# for the session holding the lock. Then issue the alter system kill session command in order to release the session holding the lock.

Note: The second session would now show the message.

Quick Reference

Context	Reference

Initialization parameters	`DML_LOCKS`
	`ENQUEUE_RESOURCES`
	`USER_DUMP_DEST`

Dynamic performance views	`V$LOCK`
	`V$LOCKED_OBJECT`
	`V$SESSION`

Data dictionary views	None

Commands	`LOCK TABLE IN lock MODE; ALTER TABLE` *`table_name`* `DISABLE TABLE LOCK;`
	`ALTER SYSTEM KILL SESSION` *`'sid,serial#'`*`;`

Packaged procedures and functions	None
Scripts	None

Lock Matrix

Type of Request	Lock Mode	Lock Target	Conflicts/Notes
Initialization parameters	None	None	No locks on reads
Lock table in Row Share mode	Mode 2	TM(RS) lock on table	Mode 6, so no exclusive DDL (this is the least restrictive lock.)
Lock table partition in Row Share mode	Mode 2 Mode 2	TM (RS)lock on table TM (RS) lock on table partition	Mode 6, so no exclusive DDL (This is the least restrictive lock.
Select for update	Mode 2 Mode 2 Mode 6	TM (RS) lock on table TM (RS) lock on each table partition TX lock on RBX TX slot	Mode 6 and any selects for update or DML on same rows No exclusive DDL

Type of Request	Lock Mode	Lock Target	Conflicts/Notes
Lock table in Row Exlusive mode	Mode 3	TM (RX) lock on table	Modes 4, 5, 6 (updates allowed, because mode 3 does not conflict with mode 3.) No share locks and no referential integrity locks
Lock table partition in Row Exclusiving mode	Mode 3 Mode 3	TM (RX) lock on table TM (RX) lock on table partition	Modes 4, 5, 6 on the same partition Updates allowed, because mode 3 does not conflict with mode 3 No share locks and no referential integrity locks
DML (up/ins/del)	Mode 3 Mode 6	TM (RX) lock on table TX lock on RBS TX slot	Modes 4, 5, 6 Select for update or DML on same rows No share locks and no referential integrity locks
DML (up/ins/del) on a partioned table	Mode 3 Mode 3 Mode 6	TM (RX) lock on table TM (RX) lock on each table partition owning the updated rows TX lock on RBS TX slot	Modes 4, 5, 6 Select for update or DML on same rows No share locks and no referential integrity locks
Lock table in Share mode	Mode 4	TM (S) lock on table	Modes 3, 5, 6 Allows Select for Update and other Share Locks No possible ORA 1555 error on locked table

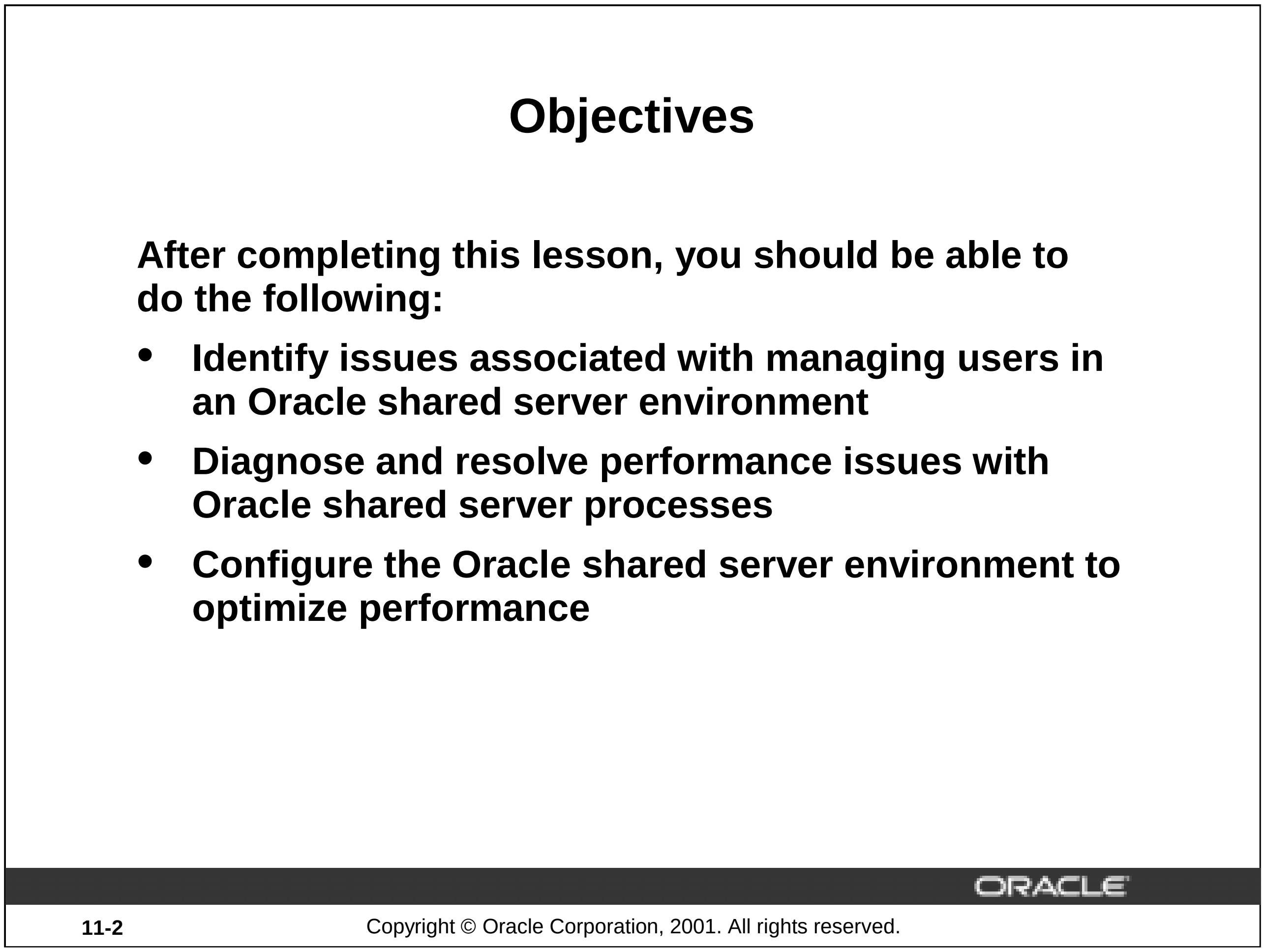
Type of Request	**Lock Mode**	**Lock Target**	**Conflicts/Notes**
Lock table partition in Share mode	Mode 2 Mode 4	TM(RS) lock on table TM(S) lock on table partition	Mode 3,5,6 on the same partition Allows Select for Update and other Share locks No possible ORA 1555 on locked table
Delete from/update to parent table with referential integrity constraint on child table, *no index on FK column in child table*, and no ON DELETE CASCADE in the FK constraint	Mode 4 Mode 3 Mode 6	TM(S) lock on child TM(RX) lock on parent TX lock on RBS TX slot	Mode 3,5,6 on child Allows Select for Update and Share lock on child No possible ORA 1555 on child Mode 4,5,6 on parent Select for update or DML on same rows against parent
Delete from/update to parent table with referential integrity constraint on child *with index on FK column in child table* and no ON DELECT CASCADE in the FK constraint	Mode 3 Mode 6	TM(RX) lock on parent TX lock on RBS TX slot	Mode 4,5,6 and any selects for update or DML on same rows against parent Updates against rows in child referred to by DML against parent
Lock table in Share Row Exclusive mode	Mode 5	TM(SRX) lock on table	Mode 3,4,5,6 Allows Select for Update only No Share locks No ORA 1555 No cascaded deletes

Type of Request	Lock Mode	Lock Target	Conflicts/Notes
Lock table in partition in Share Row Exclusive mode	Mode 2 Mode 4	TM(RS) lock on table TM(S) lock on table partition	Mode 4 on the same partition Mode 3,5,6 on any partition Allows Select for Update only No ORA 1555 No cascaded deletes
Delete from/update to parent table with referential integrity constraint on child table and *no index on FK column in child table and with ON DELETE CASCADE in the FK constraint*	Mode 5 Mode 3 Mode 6	TM(S) lock on child TM(RX) lock on parent TX lock on RBS TX slot	Mode 3,4,5,6 on child Allows Select for Update only No Share locks due to referential integrity No ORA 1555 No cascaded deletes from other parent tables that this child references Mode 4,5,6 Select for update or DML on same rows on parent
Delete from/update to parent table with referential integrity constraint on child table and *with index on FK column in child table and with ON DELETE in the FK constraint*	Mode 3 Mode 3 Mode 6	TM(RX) lock on parent TX lock on RBS TX slot	Mode 4,5,6 and any selects for update or DML on same rows against parent Mode 4,5,6 and any selects for update or DML or other cascaded deletes on same rows in child that are current targets for cascaded deletes.

Type of Request	**Lock Mode**	**Lock Target**	**Conflicts/Notes**
Lock table in Exclusive mode	Mode 6	TM(X) lock on table	Mode 2,3,4,5,6 Selects only; no DDL Most restrictive lock mode
Lock table partition in Exclusive mode	Mode 3 Mode 6	TM(X) lock on table TM(X) lock on table partition	Mode 2,3,4,5,6 on the same partition Mode 5 on any partition No exclusive DDL Most restrictive lock mode on partition
Drop, Truncate, Create Table and Create Index DDL	Mode 6 No wait	TM(X) lock on table	Mode 2,3,4,5,6 Selects only; No DDL DDL fails if any other lock mode on table due to no wait
Drop, Truncate, ADD Partition DDL	Mode 3 Mode 6 No wait	TM(X) lock on table TM(X) lock on table partition	Mode 2,3,4,5,6 on the same partition Mode 5 on any partition DDL fails if any other lock mode on table partition due to no wait

11

Tuning the Oracle Shared Server

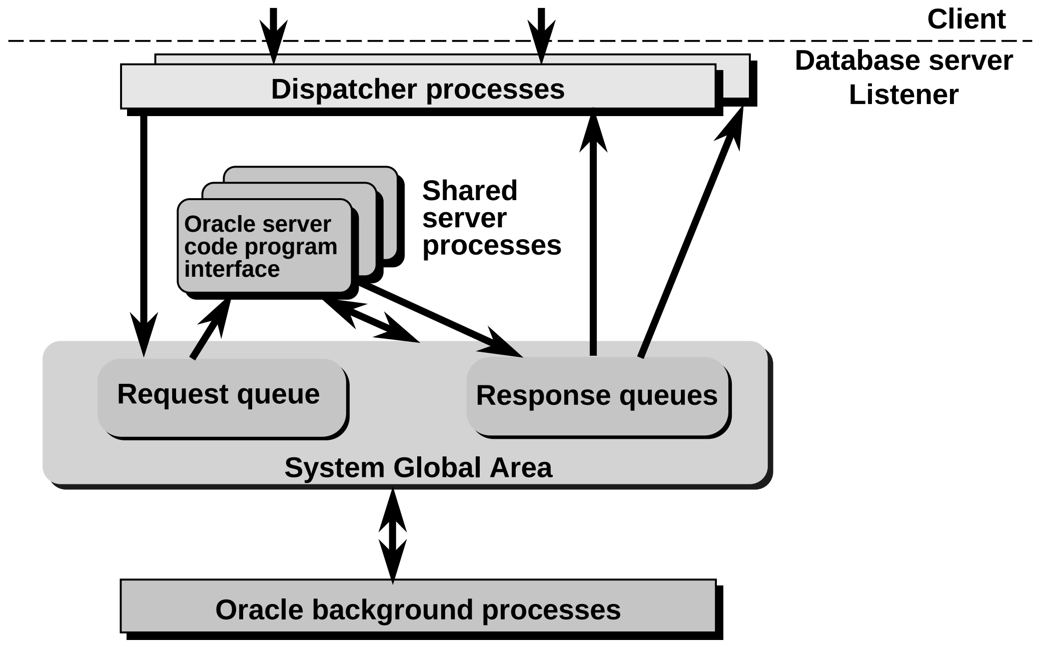


Objectives

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

- **Identify issues associated with managing users in an Oracle shared server environment**
- **Diagnose and resolve performance issues with Oracle shared server processes**
- **Configure the Oracle shared server environment to optimize performance**

Overview



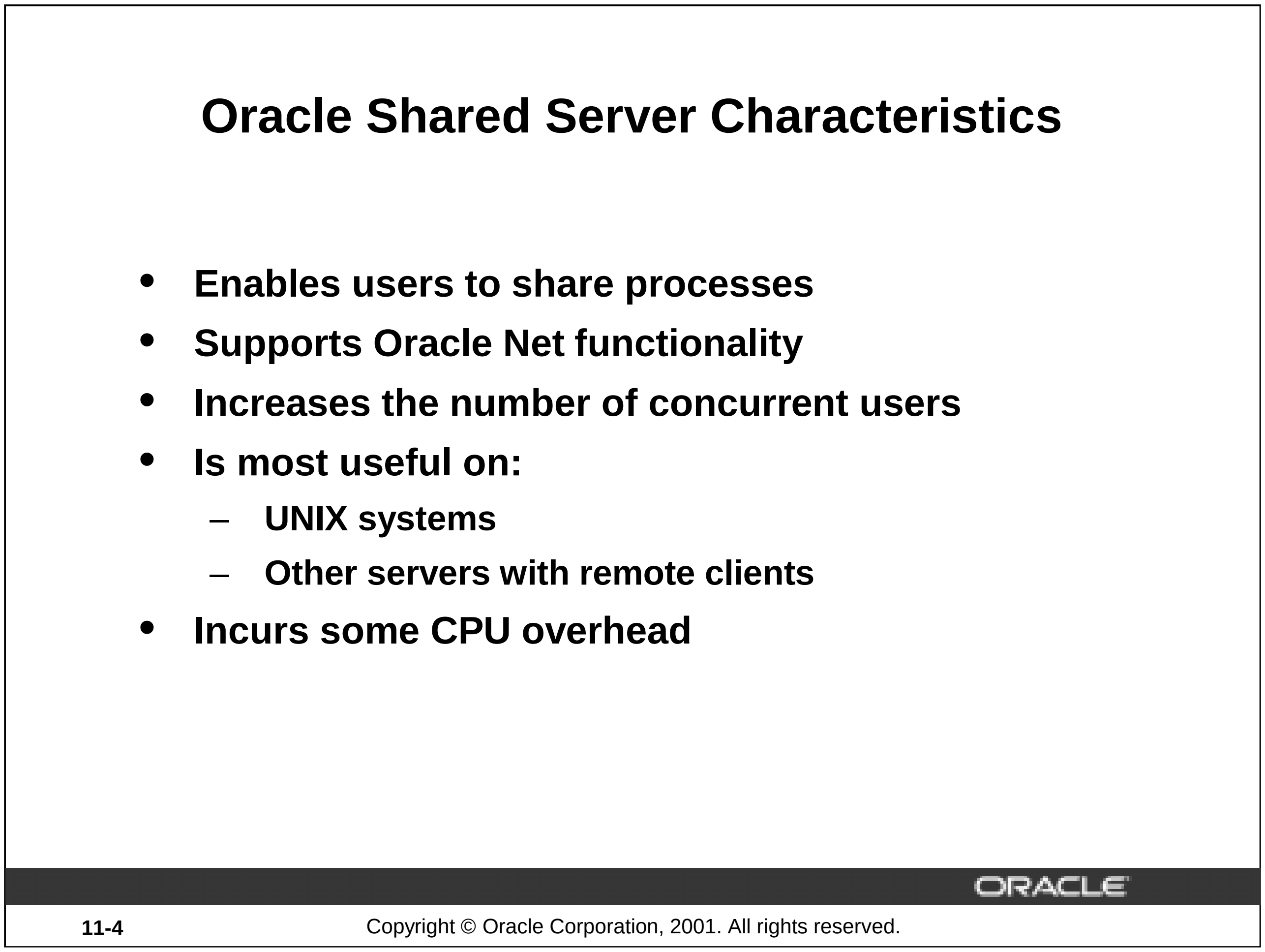
Oracle Shared Server

Oracle shared server is designed to allow multiple user processes to share a limited number of servers.

In a dedicated server environment, each user process is allocated a server process. This server process, in turn, may not be fully used by a user process, due to idle time and inactivity. However, the allocation of this server process consumes both memory and CPU resources.

When using the Oracle shared server, on the other hand, user processes are dynamically allocated to a server process that can be shared across many user processes. The dispatcher process receives a request from a user process and places it in the request queue, so that a shared server can process it and return the results to the response queue for the dispatcher. The dispatcher process returns the results to the user after the results are placed in the response queue.

A useful example is the reservations process for Oracle courses. A customer calls to inquire about a booking. The reservations clerk has a window in which to query course availability, but needs to talk to the customer for a few minutes first. The query is then sent to the database. After a bit more conversation, the customer decides which course to book, and the clerk commits the booking. In a ten-minute conversation, the dedicated server has been idle 99% of the time.



Oracle Shared Server Characteristics

- **Enables users to share processes**
- **Supports Oracle Net functionality**
- **Increases the number of concurrent users**
- **Is most useful on:**
  - **UNIX systems**
  - **Other servers with remote clients**
- **Incurs some CPU overhead**

11-4

Characteristics

The Oracle shared server provides a way for users to share processes and is meant for scaling up the number of connections that Oracle server can handle concurrently.

The Oracle shared server supports both multiplexing and pooling for user connections by means of Connection Manager and Connection Pooling.

Without shared servers, each user on a standard UNIX system or a remote client user needs a dedicated server process to access Oracle server files and memory structures. In an interactive application, where users spend much of their time talking to customers, these dedicated servers are mainly idle. They have work to do only when the user sends a query or a change to the database.

With Oracle shared servers, multiple users can share dispatcher processes, which access the Oracle Server for them. Oracle uses shared servers to process the SQL statements passed in by the dispatchers.

Oracle shared server is useful on:

- Systems that have a high overhead for a dedicated server
- Machines that are approaching limits on resources

There is no advantage in using shared servers for database-intensive work. Heavy or batch users should have dedicated servers.

Monitoring Dispatchers

- Use the following dynamic views:
  - **`V$MTS`**
  - **`V$DISPATCHER`**
  - **`V$DISPATCHER_RATE`**
- Identify contention for dispatchers by checking:
  - Busy rates
  - Dispatcher waiting time
- Check for dispatcher contention
- Add or remove dispatchers while the database is open

Identifying Contention for Dispatcher Usage

In general, dispatchers are not very busy because their task is completed quickly. However, because dispatchers are not started or stopped automatically by the server, you should monitor their usage.

You can find the limit of connections and sessions, and the current usage of sessions from the `V$MTS` view. The value for `MAXIMUM_CONNECTIONS` defaults to the value of the `SESSIONS` parameter, if `SESSIONS` is set lower than the actual limit for a dispatcher.

Query the `V$DISPATCHER` view to determine the usage for selected dispatcher processes. You identify contention for dispatchers by checking:

```
SQL> SELECT network "Protocol", status “Status”,
  2> SUM(OWNED) "Clients“,
  3> SUM(busy)*100/(SUM(busy)+SUM(idle)) "Busy Rate“
  4> FROM v$dispatcher  GROUP BY network;
```

The query returns the percentage of time the dispatcher processes of each protocol are busy. The Clients column indicates the number of clients connected using the protocol.

Guideline: In choosing the number of dispatchers, you should consider the number of clients for a dispatcher as well as the busy rate. If the busy rate of a dispatcher is over 50%, you may consider increasing the number dispatcher. You are also likely to find dispatchers for some protocols being idle. You should consider reducing the number of such idle dispatchers.

Checking Whether Users Wait for Dispatchers

You should check whether users sessions are waiting for dispatchers by executing the following query at frequent intervals:

```
SELECT DECODE(SUM(totalq),0,’No Responses’,
       SUM(wait)/SUM(totalq)) “Average wait time”
FROM   v$queue q, v$dispatcher d
WHERE  q.type    = ‘DISPATCHER’
AND    q.paddr   = d.paddr;
```

The average wait time is expressed in hundredths of a second. A steadily increasing value indicates a problem. You may want to start up more dispatchers at once if you run the query twice or more and wait times seem to be increasing.

To add or remove dispatchers, use the following command:

```
ALTER SYSTEM SET mts_dispatchers = ‘protocol, number’;
```

Allocating more dispatchers does not have any immediate effect, because users are bound to the same dispatcher until they log off. Only new connections can make use of the new dispatchers.

You can also query the `V$DISPATCHER_RATE` view to analyze contention. The `V$DISPATCHER_RATE` view contains columns grouped under `CUR`, `AVG`, and `MAX`. Compare `CUR` and `MAX` values.

If the performance of connections using shared servers are not satisfactory, and the `CUR` values are close or equal to the `MAX` values, then you should consider increasing the number of dispatchers for the corresponding protocols.

If, on the other hand, you find that the performance is satisfactory and the CUR values are substantially below the MAX values, you have configured too many dispatchers. Consider reducing the number of dispatchers.

You can use the `ALTER SYSTEM SET DISPATCHERS`... command to increase or decrease the number of dispatchers.

Monitoring Shared Servers

Oracle shared servers are started up dynamically. However, you should monitor the shared servers by:

- **Checking for shared server process contention**
- **Adding or removing idle shared servers**

Monitoring Shared Servers

Oracle shared server processes are started dynamically by the PMON background process when the existing shared servers are busy. Then, if the value of MAX_SHARED_SERVERS is higher than the number of currently available servers. Accordingly, when shared servers are idle, they are removed by PMON till the number reaches SHARED_SERVERS. Thus you do not need to monitor shared servers as closely as you should the dispatchers.

However, you may have started up more shared servers than you need by specifying a higher than required value for the SHARED_SERVERS parameter. Because Oracle9*i* Server does not terminate the unused shared server processes if they number less than that specified in SHARED_SERVERS even if they are idle, such unused shared server processes may overload the system.

You may also want to find out whether the number of servers is approaching the value of MAX_SHARED_SERVERS or (even worse) is bringing the number of processes close to the value of the PROCESSES parameter.

You can add or remove shared servers by using the following command:

```
    ALTER SYSTEM SET SHARED_SERVERS = number;
```

Monitoring Shared Servers (continued)

You should query the `V$SHARED_SERVER` view to obtain information on the current status of shared servers:

```
SELECT name, requests, busy*100/(busy+idle) “BUSY %”, status
FROM v$shared_server
WHERE status != 'QUIT';
```

You can also determine if there is contention for shared server processes by querying the `wait` and `totalq` columns of the `V$QUEUE` dynamic performance view.

Monitor these statistics occasionally while your application is running:

```
SELECT DECODE( totalq, 0, 'No Requests',
wait/totalq || ' hundredths of seconds')
"Average Wait Time Per Requests"
FROM v$queue
WHERE type = 'COMMON';
```

This query returns the total wait time for all requests and total number of requests for the request queue. The result of this query looks like this:

```
Average Wait Time per Request
-----------------------------
.090909 hundredths of seconds
```

Monitoring Process Usage

The V$CIRCUIT view displays:

- **Server address**
- **Dispatcher address**
- **User session address**

Checking Shared Connections

If a user has a problem or a process seems to be doing too much work, you may need to check on current users with shared connections.

Use the `server` column of `V$SESSION` view to ascertain the type of connections that the sessions are using.

```
SELECT SID,SERIAL#, USERNAME, SERVER
FROM   V$SESSION;
```

You can query current dispatcher and server use with the `V$CIRCUIT` view that gives you server and dispatcher addresses, and the session address for the user.

You also need to check with `V$DISPATCHER`, `V$SHARED_SERVER`, and `V$SESSION` views for the corresponding values in the `name` and `username` columns.

Shared Servers and Memory Usage

- **Some user information goes into the shared pool.**
- **Overall memory demand is lower when using shared servers.**
- **Shared servers use the user global area (UGA) for sorts.**

Using Oracle Shared Servers as Search Servers

If you decide to use the shared server, some user session information, called the user global area (UGA), is stored in the shared pool, while the data components of the session are held in the large pool. If the large pool is not configured, then all the information is stored in the shared pool.

The overall memory demand on the system decreases when using shared servers.

Shared servers use the UGA for sorts, so if you are using a shared server, you should set `SORT_AREA_RETAINED_SIZE` smaller than `SORT_AREA_SIZE`, so that memory can be released back for other users as quickly as possible.

Troubleshooting

Possible causes of problems with the shared server include the following:

- **The database listener is not running.**
- **The Oracle shared server initialization parameters are set incorrectly.**
- **The dispatcher process has been killed.**
- **The DBA does not have a dedicated connection.**
- **The `PROCESSES` parameter is too low.**

Troubleshooting

Troubleshooting the Oracle shared server environment is a key DBA function. Some common problems include the following:

- If the Oracle Net listener is not running, all attempts at shared connections fail. You need to bring it up whenever the machine is started up.
- Any Oracle Net configuration error gives a `TNS_` error message when you try to establish a shared connection.
- It is always bad practice to kill a user's server process at the operating system level (use the `ALTER SYSTEM KILL SESSION` command instead). But if a user is connected through a dispatcher, it is even more dangerous, because killing the dispatcher may affect many other users as well.
- You cannot perform privileged operations such as `STARTUP` and `SHUTDOWN` using a shared server. Make sure you have a dedicated connection for yourself as DBA.
- Your servers and dispatchers count as background processes for the instance. Be careful that your setting of `PROCESSES` allows for all possible servers and dispatchers, or new users may not be able to log in. Setting `MTS_MAX_SERVERS` and `MTS_MAX_DISPATCHERS` can act as a useful ceiling.
- If the parameters (`INSTANCE_NAME`, `SERVICE_NAMES` or `DB_DOMAIN`) are not set or if they are set to an incorrect value, then the automatic instance registration will not work.

Obtaining Dictionary Information

Dynamic performance views:

V$CIRCUIT

V$DISPATCHER

V$DISPATCHER_RATE

V$QUEUE

V$MTS

V$SESSION

V$SHARED_SERVER

Using Dynamic Performance Views

You can use different dynamic performance views to obtain information about the Oracle shared server environment. Use the Oracle8*i* reference manual to obtain details about each of the following views:

- `V$CIRCUIT`: Contains information about user connections to the database
- `V$DISPATCHER`: Provides information on the dispatcher processes
- `V$DISPATCHER_RATE`: Provides rate statistics for the dispatcher processes
- `V$QUEUE`: Contains information on the multithread message queues
- `V$MTS`: Contains information for tuning the Oracle shared server
- `V$SESSION`: Lists session information for each current session
- `V$SHARED_SERVER`: Contains information about the shared server processes

A query to report the dispatcher, session, and process mapping using shared servers:

```
SELECT   d.network network, d.name disp, s.username oracle_user,
         s.sid sid,s.serial# serial#, p.username os_user,
         p.terminal terminal, s.program program
FROM     v$dispatcher d, v$circuit c, v$session s, v$process p
WHERE    d.paddr = c.dispatcher(+)
AND      c.saddr = s.saddr(+)
AND      s.paddr = p.addr (+)
order by d.network, d.name, s.username
```

Summary

In this lesson, you should have learned how to:

- Describe the Oracle shared server as a resource-sharing configuration
- List some situations in which it is appropriate to use the Oracle shared server
- Monitor dispatcher and server usage
- Troubleshoot Oracle shared server configuration

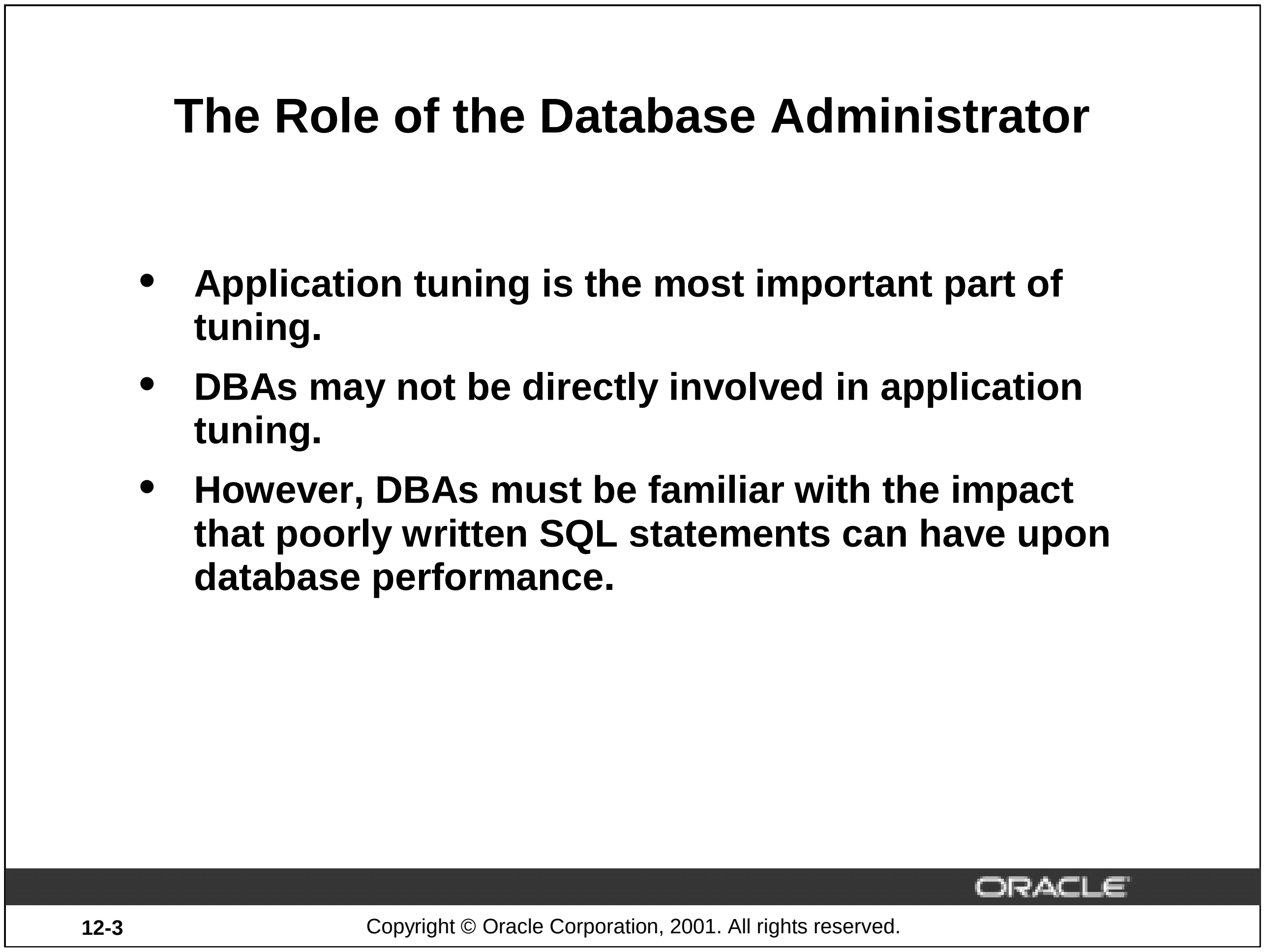
12

Application Tuning

Objectives

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

- The role of the DBA in tuning applications
- Different storage structures, and why one storage structure might be preferred over another
- The different types of indexes
- Index-organized tables
- Materialized views and the use of query rewrites
- Requirements for OLTP, DSS, and hybrid systems



The Role of the Database Administrator

- **Application tuning is the most important part of tuning.**
- **DBAs may not be directly involved in application tuning.**
- **However, DBAs must be familiar with the impact that poorly written SQL statements can have upon database performance.**

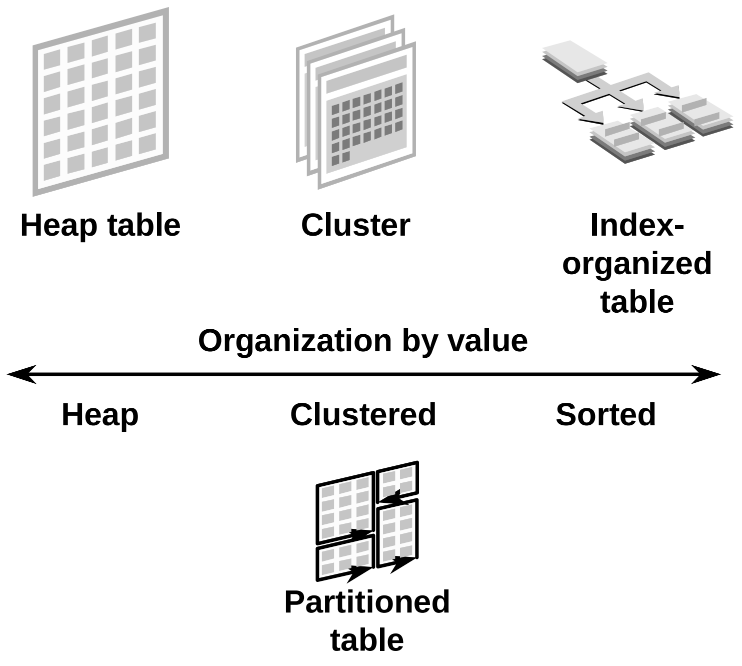
12-3

Database Administrator Tasks

Application design and application tuning provide the greatest performance benefits. However, the method in which data is selected and the amount of data that is selected have serious implications for application performance if the statements that execute those operations are not properly written.

As a database administrator (DBA), you may not be directly involved in the tuning of an application; application developers are usually responsible for developing applications and writing SQL statements. However, as a DBA you need to be aware of the impact that poorly written SQL statements can have upon the database environment. You should be able to provide assistance with application tuning and identify inefficient SQL statements. You should also be aware of optimizer plan stability, implemented through stored outlines, and query rewrites using materialized views and dimensions.

Data Storage Structures



Selecting the Physical Structure

Factors affecting the selection:

- **Rows read in groups**
- **`SELECT` or DML statements**
- **Table size**
- **Row size, row group, and block size**
- **Small or large transactions**
- **Using parallel queries to load or for `SELECT` statements**

Selecting the Physical Structure

The DBA's goal is to enable reads and writes to happen as fast as possible. In order to achieve this goal, the following factors must be taken into account:

Rows read in groups

If the application uses rows in groups, then investigate storing the rows in clusters. Also, because clusters do not perform well with high DML activity, you need to look at the amount of DML activity as SELECT or DML statements.

In order to distribute the workload among the disks and controllers, you should know what tables are going to have a high number of reads, and which tables will have a high number of writes. Avoid having highly accessed tables on the same drive or controller.

Because queries can hit any portion of the table, look for specific *hot-spots*. DML is more likely to have the active extent as the *hot spot*.

Table size

Consider using a separate tablespace for large tables. For very large partitioned tables, multiple tablespaces can be used. This arrangement assists in managing and distributing the workload evenly among disks.

Selecting the Physical Structure (continued)

Row Size, Row Groups, and Block Size

Oracle9*i* allows multiple block sizes within the same database; thus tables that have a larger row size can have a larger block size. A larger block size can also assist if the application uses full table scans, or if the table is clustered.

Small or Large Transactions

Small transactions require less undo space to store rollback information. Larger transactions require more undo space. Because larger transactions add more data, more available free space is required within the tablespace.

Parallel Queries

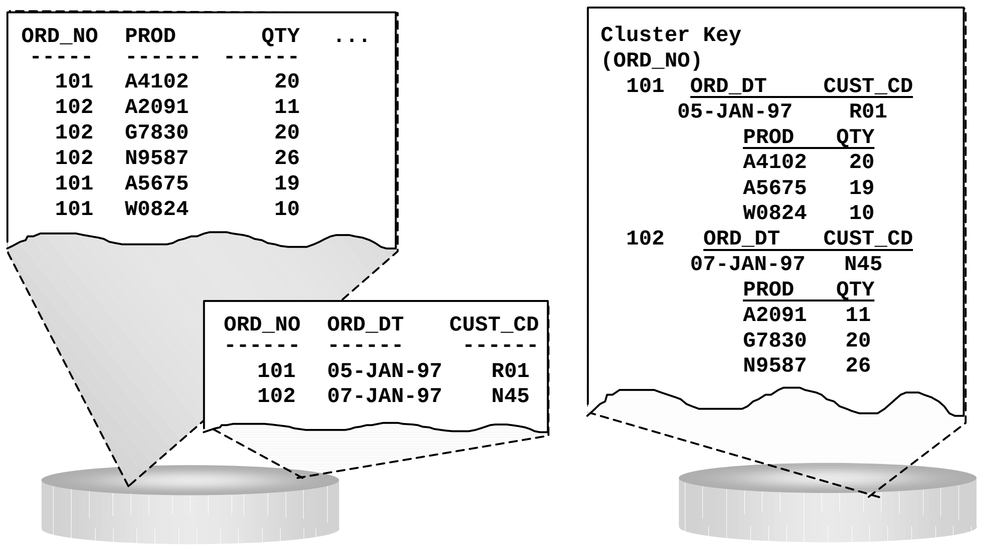
If large queries are occurring, you should investigate having multiple server processes distribute the workload between them. Using multiple server processes to distribute the table between disks and controllers will lessen contention and therefore enhance performance.

Data Access Methods

To enhance performance, you can use the following data access methods:

- **Clusters**
- **Indexes**
  - **B-tree (normal or reverse key)**
  - **Bitmap**
  - **Function based**
- **Index-organized tables**
- **Materialized views**

Clusters



```
ORD_NO  PROD       QTY  ...
-----   ------  ------
   101  A4102       20
   102  A2091       11
   102  G7830       20
   102  N9587       26
   101  A5675       19
   101  W0824       10
```

```
ORD_NO  ORD_DT     CUST_CD
------  ------     ------
   101  05-JAN-97    R01
   102  07-JAN-97    N45
```

```
Cluster Key
(ORD_NO)
  101  ORD_DT     CUST_CD
      05-JAN-97     R01
           PROD    QTY
           A4102    20
           A5675    19
           W0824    10
  102   ORD_DT    CUST_CD
       07-JAN-97   N45
           PROD    QTY
           A2091   11
           G7830   20
           N9587   26
```

Unclustered `orders` and `order_item` tables

Clustered `orders` and `order_item` tables

Definition of Clusters

A cluster is a group of one or more tables that share the same data blocks because they share common columns and are often used together in join queries. Storing tables in clusters allows a DBA to denormalize data that is transparent to the end user and programmer.

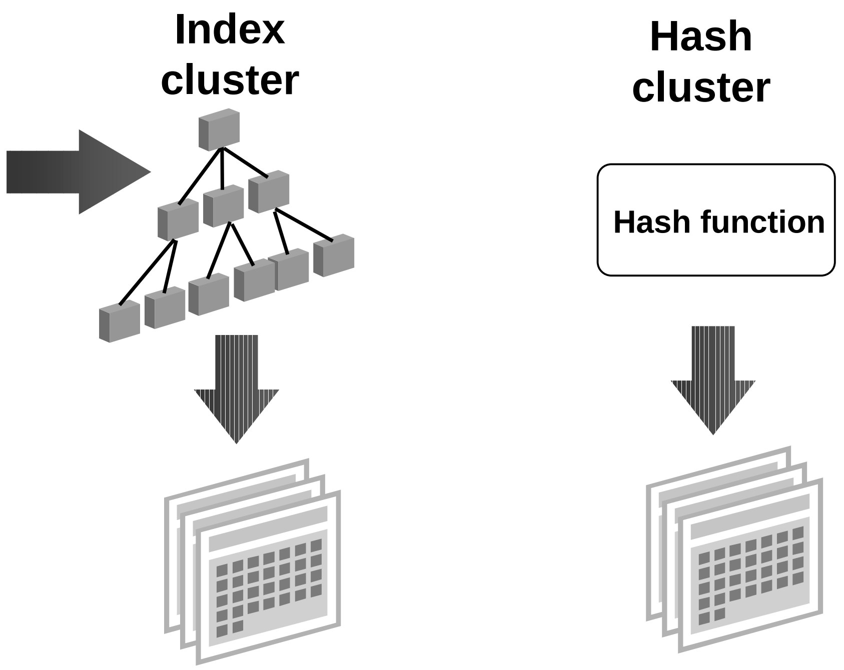
Performance Benefits of Clusters

- Disk I/O is reduced and access time improved for joins of clustered tables.
- Each cluster key value is stored only once for all the rows of the same key value; it therefore, uses less storage space.

Performance Consideration

Full table scans are generally slower on clustered tables than on nonclustered tables.

Cluster Types



Index Clusters

An index cluster uses an index, known as the *cluster index,* to maintain the data within the cluster. The cluster index must be available to store, access, or maintain data in an index cluster.

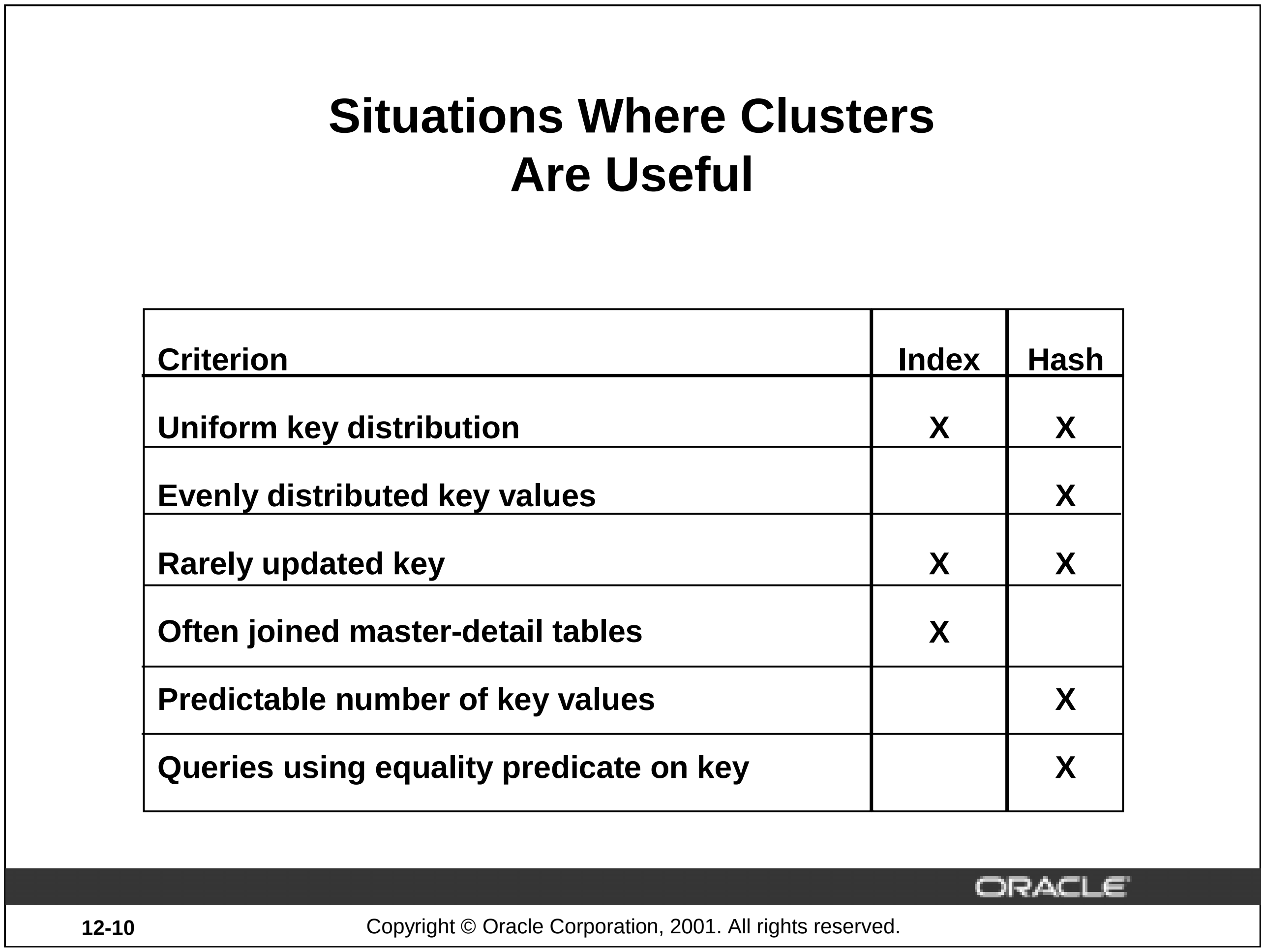
The cluster index is used to point to the block that contains the rows with a given key value. The structure of a cluster index is similar to that of a normal index.

Although a normal index does not store null key values, cluster indexes store null keys. There is only one entry for each key value in the cluster index. Therefore, a cluster index is likely to be smaller than a normal index on the same set of key values.

Hash Clusters

A hash cluster uses a hash algorithm (either user-defined or system-generated) to calculate the location of a row, both for retrieval and for DML operations.

For equality searches that use the cluster key, a hash cluster can provide greater performance gains than an index cluster, because there is only one segment to scan (no index access is needed).



Situations Where Clusters Are Useful

Criterion	Index	Hash
Uniform key distribution	X	X
Evenly distributed key values		X
Rarely updated key	X	X
Often joined master-detail tables	X	
Predictable number of key values		X
Queries using equality predicate on key		X

12-10

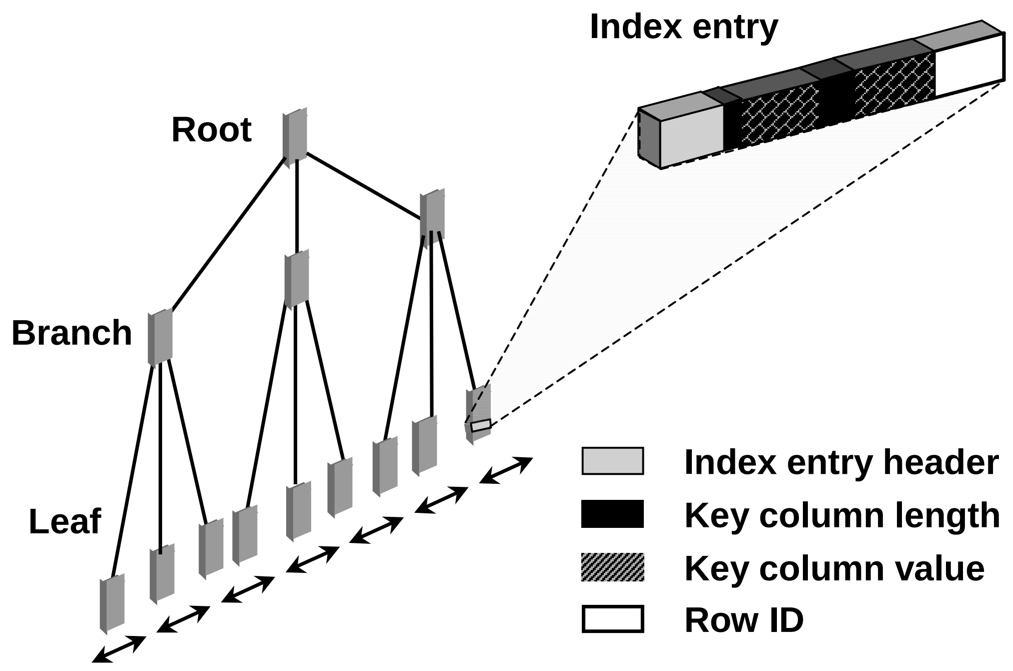
When Not to Use Clusters

- If a full scan is executed often on one of the clustered tables: This table is stored on more blocks than if it had been created alone.
- If the data for all rows of a cluster key value exceeds one or two Oracle blocks: In order to access an individual row in a clustered key table, the Oracle server reads all blocks containing rows with the same value.

When Not to Use Hash Clusters

- If the table is constantly growing and if it is impractical to rebuild a new, larger hash cluster
- If your application often performs full table scans and you had to allocate a great deal of space to the hash cluster in anticipation of the table growing

B-Tree Indexes



When to Create B-Tree Indexes

B-tree indexes typically improve the performance of queries that select a small percentage of rows from a table. As a general guideline, you should create indexes on tables that are often queried for less than 5% of the table’s rows. This value may be higher in situations where all data can be retrieved from an index, or where the indexed columns can be used for joining to other tables.

How Indexes Grow

Indexes are always balanced, and they grow from the bottom up.

As rows are added, the leaf block fills. When the leaf block is full, the Oracle server splits it into two blocks and puts 50% of the block's contents into the original leaf block, and 50% into a new leaf block.

Because another block has been added to the index, this newly added block must be added to the directory entry in the parent branch block. If this parent branch block is full, the parent branch block is split in a similar way to the leaf block, with 50% of the existing contents being divided between the existing and new branch blocks. If required, this pattern is repeated until the place where the root block becomes a branch block, and a new root block is added.

The more levels an index has, the less efficient it may be. Additionally, an index with many rows deleted might not be efficient. Typically, if 15% of the index data is deleted, then you should consider rebuilding the index.

How to Solve B-Tree Index Performance Degradation

You should rebuild your indexes regularly. However, this can be a time-consuming task, especially if the base table is very large. In Oracle9*i*, you can create and rebuild indexes online, and parallelization is possible as well. While the index is being rebuilt, the associated base table remains available for queries and DML operations. You can also compute statistics for the cost-based optimizer while rebuilding the index.

```
SQL> ALTER INDEX i_name REBUILD ONLINE;
```

The `ONLINE` keyword specifies that DML operations on the table or partition are allowed during rebuilding of the index.

Restriction: Parallel DML is not supported during online index building. If you specify `ONLINE` and then issue parallel DML statements, an error is returned.

Compressed Indexes

- **When creating the index:**

```
SQL> create index emp_last_name_idx
  2  on employees (last_name, first_name)
  3  compress;
```

- **When rebuilding the index:**

```
SQL> alter index emp_last_name_idx
  2  rebuild compress;
```

- **Specify `NOCOMPRESS` as the default to disable key compression.**

Key Compression

Specify `COMPRESS` to enable key compression, which eliminates repeated occurrence of key column values and may substantially reduce storage. Use integer to specify the prefix length (number of prefix columns to compress). For unique indexes, the valid range of prefix length values is from 1 to the number of key columns minus 1 (the default value).

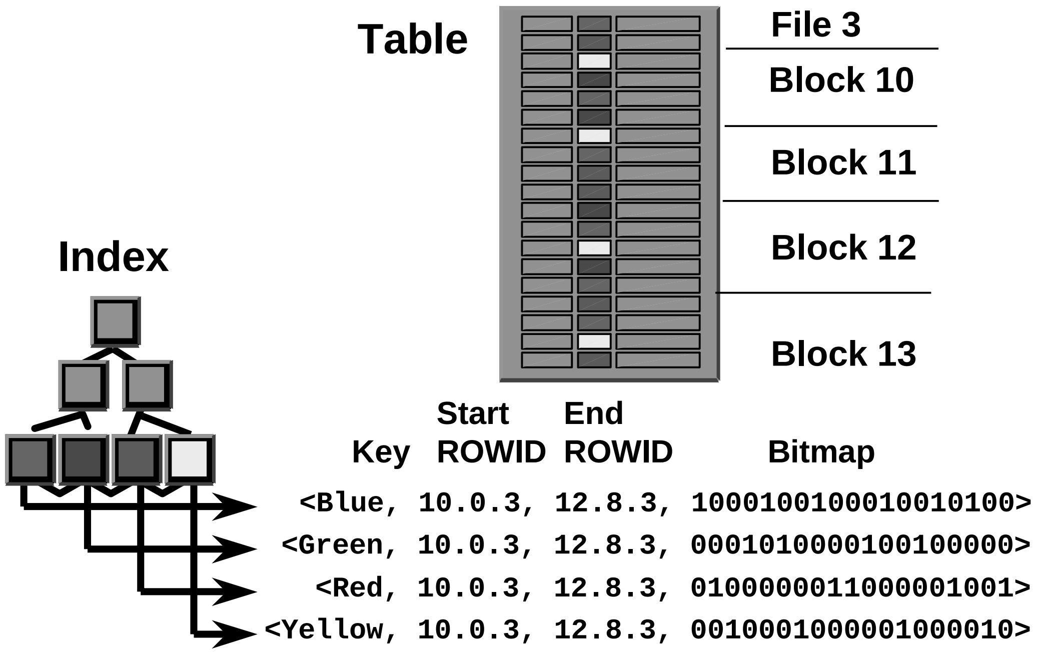
For nonunique indexes, the valid range of prefix length values is from 1 to the number of key columns. The default prefix length is the number of key columns.

Key compression is useful in many different scenarios, such as:

- In a nonunique regular index, Oracle stores duplicate keys with the row ID appended to the key to break the duplicate rows. If key compression is used, Oracle stores the duplicate key as a prefix entry on the index block without the row ID. The rest of the rows are suffix entries that consist of only the row ID.
- This same behavior can be seen in a unique index that has a key of the form (item, time stamp), for example, `stock_ticker`, or `transaction_time`. Thousands of rows can have the same `stock_ticker` value, with `transaction_time` preserving uniqueness. On a particular index block, a `stock_ticker` value is stored only once as a prefix entry. Other entries on the index block are `transaction_time` values stored as suffix entries that refer to the common `stock_ticker` prefix entry.

In some cases, however, key compression cannot be used. For example, in a unique index with a single attribute key, key compression is not possible because there is a unique piece, but there are no grouping pieces to share.

Bitmap Indexes



```
 <Blue, 10.0.3, 12.8.3, 1000100100010010100>
<Green, 10.0.3, 12.8.3, 0001010000100100000>
  <Red, 10.0.3, 12.8.3, 0100000011000001001>
<Yellow, 10.0.3, 12.8.3, 0010001000001000010>
```

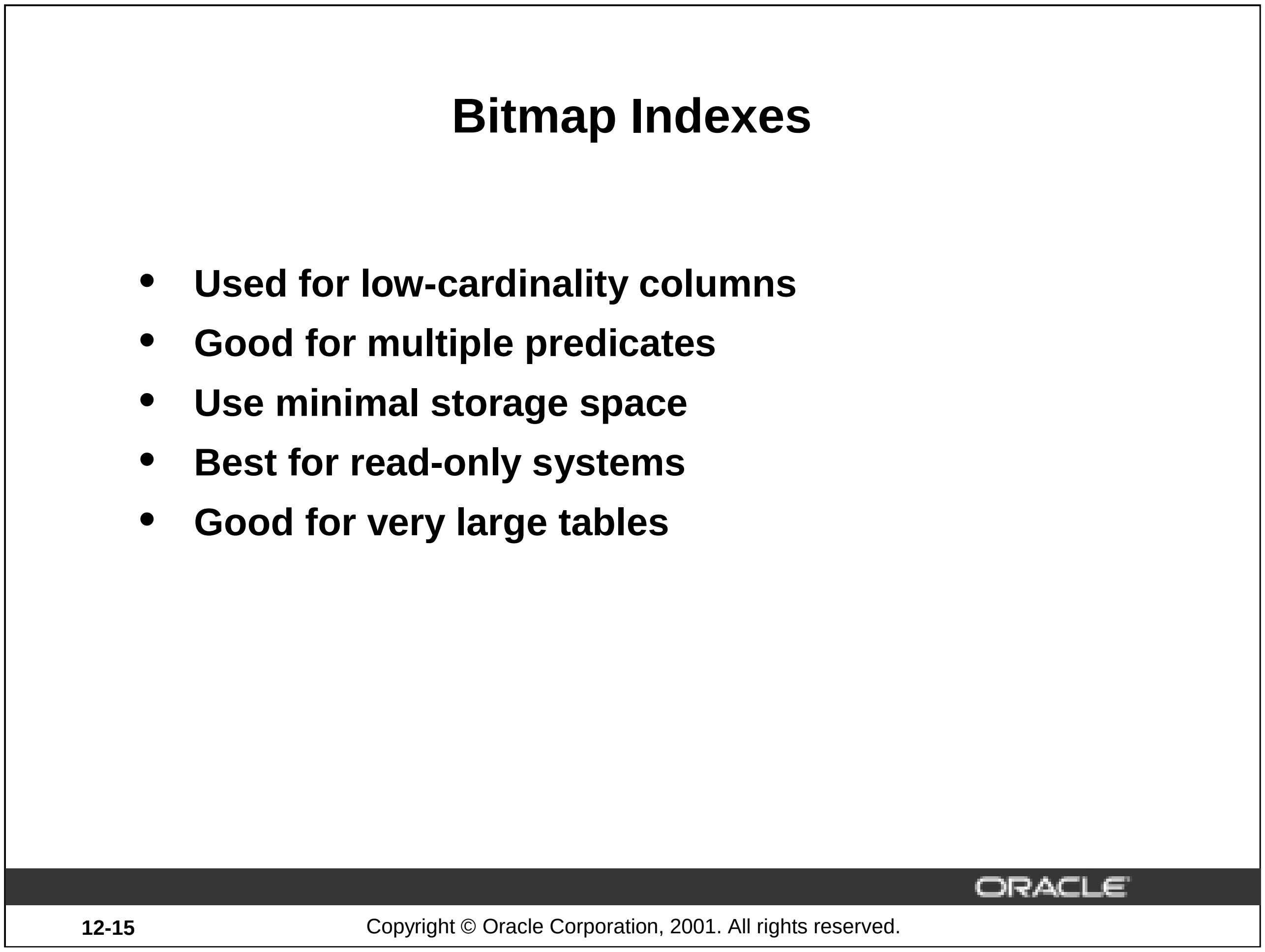
12-14 Copyright © Oracle Corporation, 2001. All rights reserved.

Bitmap Indexes

With bitmap indexes, you can store data that has few distinct values within the column, such as `gender` or `job_title`. The index consists of a range of rows stored in the index, and then a *map* of binary values for each key. The value is “on”, that is 1, if the key is true for that row. In the example on the slide, the value in the bitmap is on for only one color. The row item is blue, then the value is a 1 for blue, and 0 for all other combinations.

Bitmap indexes perform best when there are few variations for the value, but millions of rows. Bitmap indexes also help resolve Boolean type constraints; for example, if the user requires all items that are blue and yellow, or if items colored green or red are wanted.

DML statements do not perform well with bitmap indexes, so for high DML activity, do not use a bitmap index.



Bitmap Indexes

- Used for low-cardinality columns
- Good for multiple predicates
- Use minimal storage space
- Best for read-only systems
- Good for very large tables

ORACLE

12-15

When to Create Bitmapped Indexes

- Bitmap indexes are intended for *low-cardinality* columns that contain a limited number of values.
- If you use a query with multiple `WHERE` conditions, the Oracle server can use logical bit-AND or bit-OR operations to combine this bitmap with bitmaps for other columns.

Performance Considerations

- Bitmap indexes use little storage space: one entry per distinct key value, stored in a compressed form. Each bitmap is divided into bitmap segments (up to one-half block).
- They work very fast with multiple predicates on low-cardinality columns.
- They are particularly suited to large, read-only systems such as decision support systems.
- DML statements slow down performance:
  - They are not suited for OLTP applications.
  - Locking is at the bitmap-segment, not entry, level.
- Bitmap indexes store null values, B*Tree indexes do not.
- Parallel query, parallel data manipulation language (PDML), and parallelized `CREATE` statements work with bitmap indexes.

Creating and Maintaining Bitmap Indexes

```
SQL> create BITMAP INDEX departments_idx
  2    on departments(manager_id)
  3    storage   (initial 200k  next 200k
  4    pctincrease 0 maxextents 50)
  5*   tablespace indx;
```

Maintenance Considerations

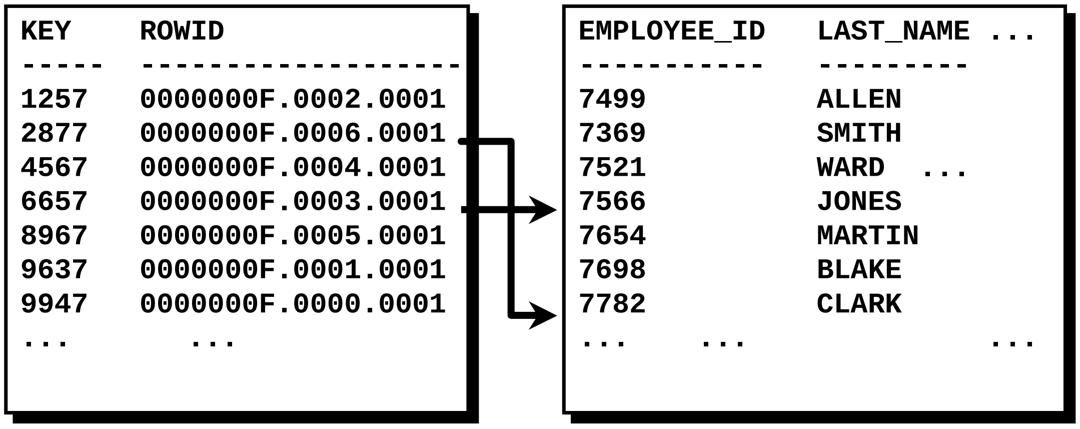
In a data warehousing environment, data is usually maintained by way of bulk inserts and updates. Index maintenance is deferred until the end of each DML operation. For example, if you insert 1,000 rows, then the inserted rows are placed into a sort buffer, and then the updates of all 1,000 index entries are batched. (This is why `SORT_AREA_SIZE` must be set properly for good performance with inserts and updates on bitmap indexes.) Thus, each bitmap segment is updated only once per DML operation, even if more than one row in that segment changes.

In Oracle9*i*, there are different parameters affecting the sort area depending on the value of `WORKAREA_SIZE_POLICY`. See the chapter titled “Optimizing Sort Operations” for more details.

B-Tree Indexes and Bitmap Indexes

B-Tree indexes	Bitmap indexes
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 AND / OR predicates	Efficient for queries using AND / OR predicates
Row-level locking	Bitmap segment-level locking
More storage	Less storage
Useful for OLTP	Useful for DSS

Reverse Key Index



```
KEY    ROWID
-----  ------------------
1257   0000000F.0002.0001
2877   0000000F.0006.0001
4567   0000000F.0004.0001
6657   0000000F.0003.0001
8967   0000000F.0005.0001
9637   0000000F.0001.0001
9947   0000000F.0000.0001
...       ...
```

```
EMPLOYEE_ID   LAST_NAME ...
-----------   ---------
7499          ALLEN
7369          SMITH
7521          WARD  ...
7566          JONES
7654          MARTIN
7698          BLAKE
7782          CLARK
...    ...              ...
```

Index on `employee_id` column

`Employees` table

Definition

Creating a reverse key index reverses the bytes of each column key value, keeping the column order in case of a composite key.

When to Create Reverse Key Indexes

An ever-increasing key (such as sequential numbers for invoices, employees, or order numbers) repetitively degrades the index height (BLEVEL statistic); you can avoid forced block splits by spreading the index entries more evenly.

Disadvantage

When application statements specify ranges, the explain plan produces a full table scan, because the index is not usable for a range scan.

Creating Reverse Key Indexes

```
SQL> create unique index i1_t1 ON t1(c1)
  2  REVERSE pctfree 30
  3  storage(initial 200k  next 200k
  4          pctincrease 0 maxextents 50)
  5  tablespace indx;
```

```
SQL> create unique index i2_t1 ON t1(c2);
SQL> alter index i2_t1 REBUILD REVERSE;
```

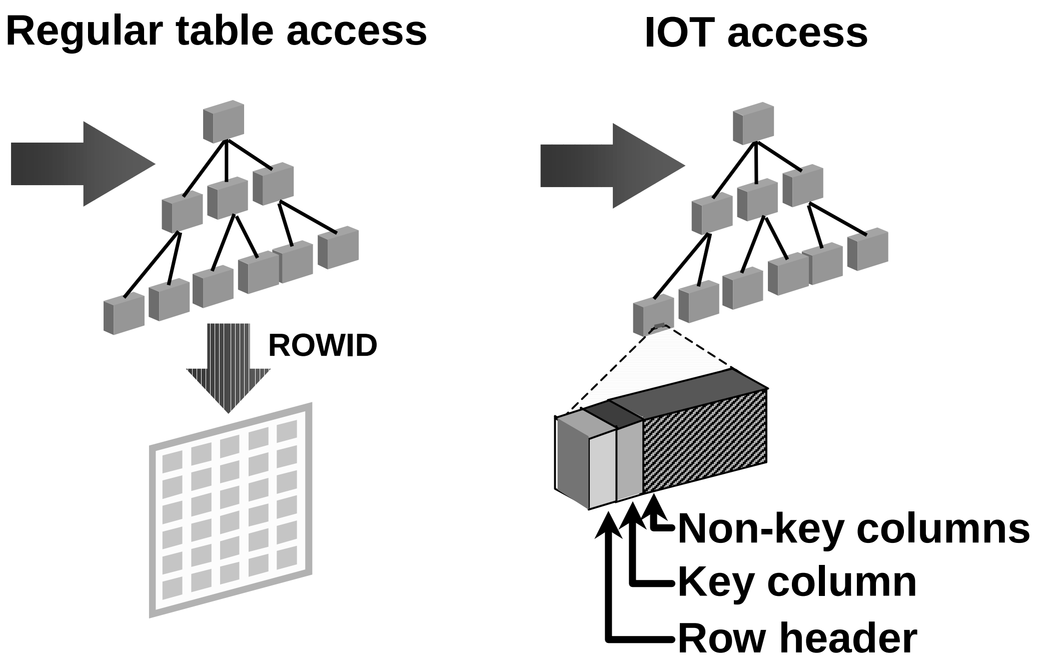
12-19

Identifying Reverse Key Indexes

Use the following query to list the names of all reverse key indexes:

```
SQL>select index_name, index_type
  2  from dba_indexes
  3  where   index_type like '%REV';
INDEX_NAME        INDEX_TYPE
--------------------------
I2_T1             NORMAL/REV
```

Index-Organized Tables (IOTs)



Definition

An index-organized table is like a regular table with an index on one or more of its columns, but instead of maintaining two separate segments for the table and the B-tree index, the database system maintains one single B-tree structure that contains both the primary key value and the other column values for the corresponding row.

Index-organized tables are suitable for frequent data access through the primary key, or through any key that is a prefix of the primary key, such as in applications that use inverted indexes. These indexes keep the value and all its locations together. Each word has one entry, and that entry records all the places in which the word occurs, and therefore the index could be used to recreate the document. These indexes are used in Intermedia.

For index-organized tables, a primary key constraint is mandatory.

Benefits

- No duplication of the values for the primary key column (index and table columns in indexed tables), therefore less storage is required
- Faster key-based access for queries involving exact match or range searches, or both.

Index-Organized Tables and Heap Tables

Compared to heap tables, IOTs have:

- **Faster key-based access to table data**
- **Reduced storage requirements**
- **Secondary indexes and logical row IDs**
- **The following restrictions:**
  - **Must have a primary key**
  - **Cannot use unique constraints**
  - **Cannot be clustered**

12-21

Logical Row IDs

Index-organized tables do not have regular (physical) row IDs. However, the concept of logical row IDs was introduced to overcome certain restrictions caused by the lack of physical row IDs. Logical row IDs give the fastest possible access to rows in IOTs by using two methods:

- A physical *guess* whose access time is equal to that of physical row IDs
- Access without the guess (or after an incorrect guess); this performs similar to a primary key access of the IOT

The guess is based on knowledge of the file and block that a row resides in. The latter information is accurate when the index is created, but changes if the leaf block splits. If the guess is wrong and the row no longer resides in the specified block, then the remaining portion of the logical row ID entry, the primary key, is used to get the row.

Logical row IDs are stored as a variable-length field, where the size depends on the primary key value being stored.

The `UROWID` data type enables applications to use logical row IDs in the same way they use row IDs, for example, selecting row IDs for later update or as part of a cursor. `UROWID` can also be used to store row IDs from other databases, accessed by way of gateways. Finally the `UROWID` type can also be used to reference physical row IDs.

Creating Index-Organized Tables

```
SQL> CREATE TABLE countries
    ( country_id      CHAR(2)
      CONSTRAINT  country_id_nn NOT NULL,
      country_name    VARCHAR2(40),
      currency_name   VARCHAR2(25),
      currency_symbol VARCHAR2(3),
      map             BLOB,
      flag            BLOB,
      CONSTRAINT     country_c_id_pk
        PRIMARY KEY (country_id))
    ORGANIZATION INDEX
    PCTTHRESHOLD 20
    OVERFLOW TABLESPACE USERS;
```

ORACLE

Creating Index-Organized Tables

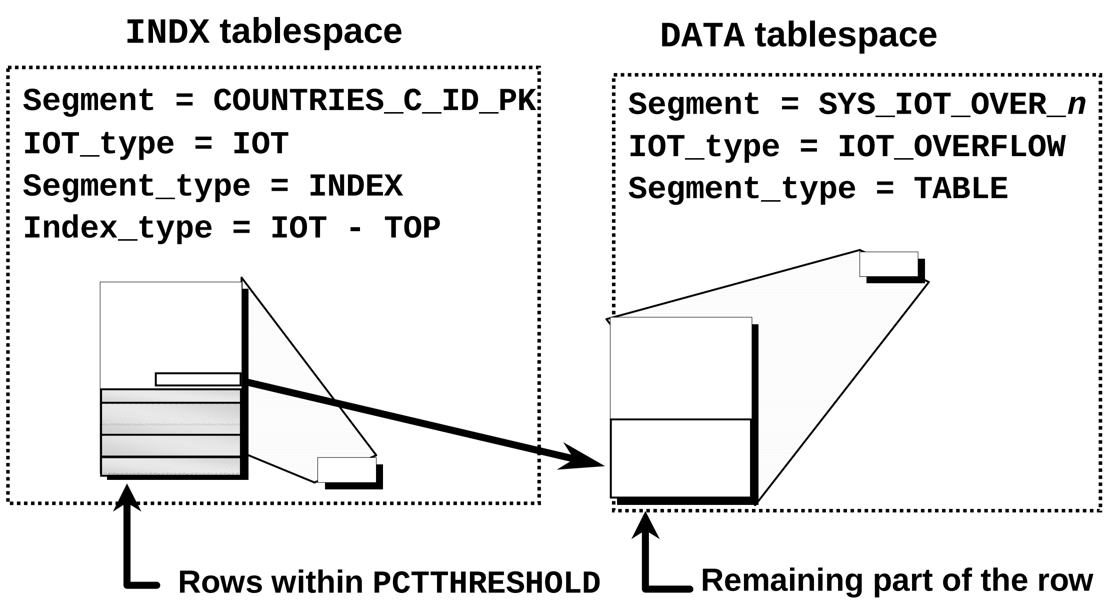
Regular B-tree index entries are usually small. They consist of only the primary key value and a `row ID` value for each row. Many of these small index entries can be stored in a leaf block. This is not necessarily the case for index-organized tables, because they store full rows, or at least as much as fits without exceeding the limit set by `PCTTHRESHOLD`.

Storing large entries in index leaf blocks slows down index searches and scans. You can specify that the rows go into an overflow area, by setting a threshold value that represents a percentage of block size.

Of course, the primary key column must always be stored in the IOT index blocks as a basis for searching. But you can keep nonkey values in a separate area, the row overflow area, so that the B-tree itself remains densely clustered.

The three clauses that influence IOT row overflow are discussed on the next page.

IOT Row Overflow



The `PCTTHRESHOLD` Clause

This clause specifies the percentage of space reserved in the index block for an index-organized table row. If a row exceeds the size calculated based on this value, all columns after the column named in the `INCLUDING` clause are moved to the overflow segment. If `OVERFLOW` is not specified, then rows exceeding the threshold are rejected. `PCTTHRESHOLD` defaults to 50 and must be a value from 0 to 50.

The `INCLUDING` Clause

This clause specifies a column at which to divide an index-organized table row into index and overflow portions. The server accommodates all nonkey columns up to the column specified in the `INCLUDING` clause in the index leaf block, provided it does not exceed the specified threshold.

The `OVERFLOW` Clause and Segment

This clause specifies that index-organized table data rows exceeding the specified threshold are placed in the data segment defined by the segments attributes, which specify the tablespace, storage, and block utilization parameters.

IOT Dictionary Views

```
SQL> select table_name,tablespace_name,iot_name,iot_type
  2  from   DBA_TABLES;
TABLE_NAME         TABLESPACE_NAME  IOT_NAME  IOT_TYPE
-----------------  ---------------  --------  ------------
COUNTRIES                                              IOT
SYS_IOT_OVER_2268  USER_DATA        COUNTRIES IOT_OVERFLOW
```

```
SQL> select index_name,index_type,tablespace_name,table_name
  2  from   DBA_INDEXES;
INDEX_NAME         INDEX_TYPE  TABLESPACE  TABLE_NAME
-----------------  ----------  ----------  ----------
COUNTRIES_C_ID_PK  IOT - TOP   INDX        COUNTRIES
```

```
SQL> select segment_name,tablespace_name,segment_type
  2  from   DBA_SEGMENTS;
SEGMENT_NAME        TABLESPACE  SEGMENT_TYPE
-----------------   ----------  ------------
SYS_IOT_OVER_2268        DATA        TABLE
COUNTRIES_C_ID_PK        INDX        INDEX
```

DBA_TABLES

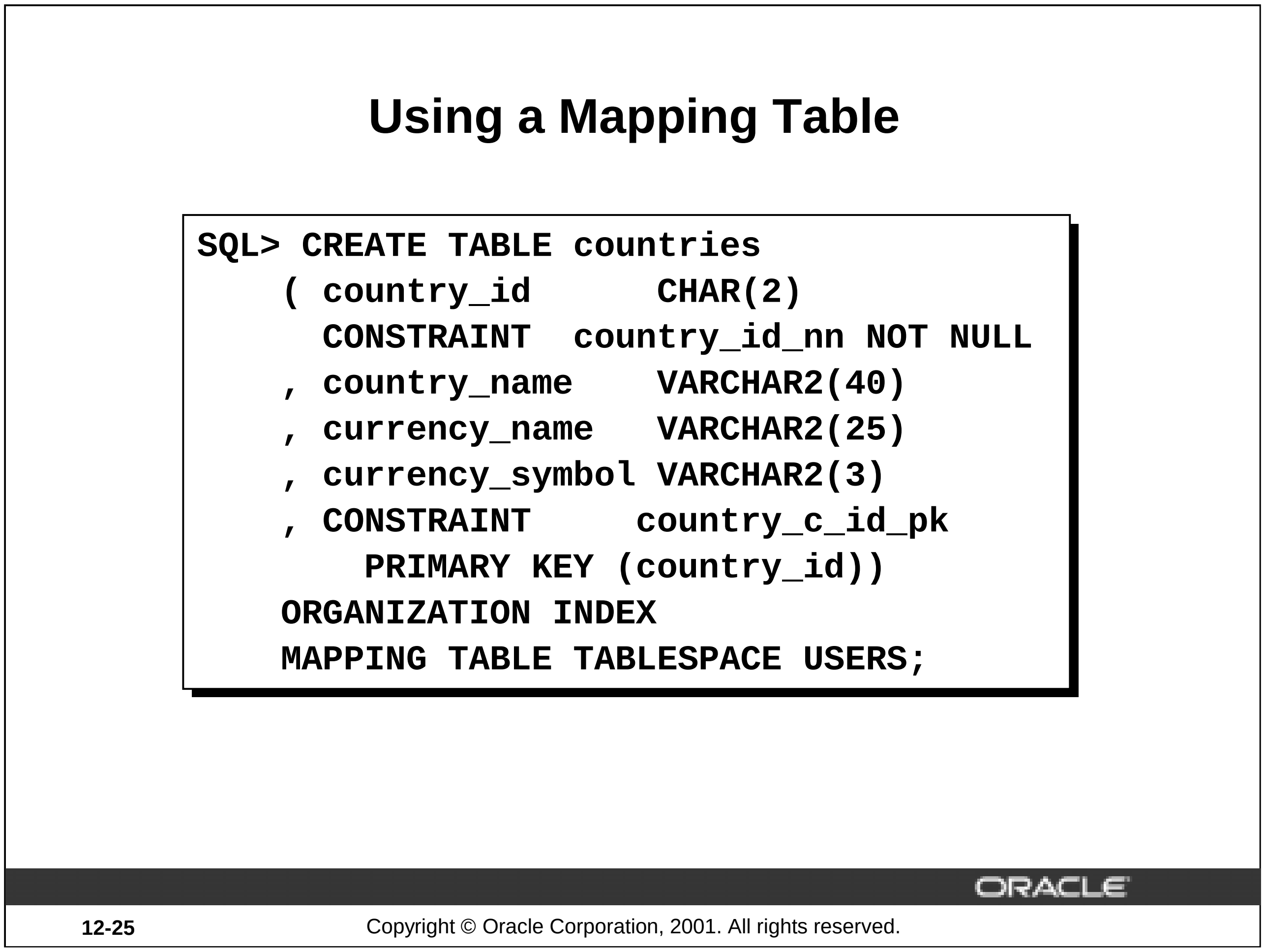
The IOT_TYPE and IOT_NAME columns provide information about index-organized tables. If there is no overflow area for an IOT, then there is only a single dictionary entry. The value of IOT_TYPE is IOT, and the IOT_NAME field is blank. If an overflow area has been specified, then its name appears as an additional new table in the list of table names. The overflow table is called SYS_IOT_OVER_*nnnn* (where *nnnn* is the object ID of the table segment) in DBA_TABLES. IOT_TYPE is set to IOT_OVERFLOW for this table, and IOT_NAME is set to the name of the index-organized table to which it belongs.

DBA_INDEXES

The IOT table name listed is the same as that used in the CREATE TABLE command, but an index name is also listed for the table, with a default name of SYS_IOT_TOP_*nnnn*.

DBA_SEGMENTS

The name of the overflow segment is listed here with the same name as mentioned above, SYS_IOT_TOP_*nnnn*. The table itself is listed as SYS_IOT_TOP_*nnnn*.



Using a Mapping Table

```
SQL> CREATE TABLE countries
    ( country_id      CHAR(2)
      CONSTRAINT  country_id_nn NOT NULL
    , country_name    VARCHAR2(40)
    , currency_name   VARCHAR2(25)
    , currency_symbol VARCHAR2(3)
    , CONSTRAINT     country_c_id_pk
        PRIMARY KEY (country_id))
    ORGANIZATION INDEX
    MAPPING TABLE TABLESPACE USERS;
```

12-25

Using a Mapping Table

Oracle9*i* allows a bitmap index to be built on an index-organized table. In order to allow this there has to be a mapping table.

A bitmap index on an index-organized table is similar to a bitmap index on a heap table, except that the row IDs used in the bitmap index on an index-organized table are those of a mapping table and not the base table. The mapping table maintains a mapping of logical row IDs (needed to access the index-organized table) to physical row IDs (needed by the bitmap index code). There is one mapping table per index-organized table and it is used by all the bitmap indexes created on that index-organized table.

In a heap organized base table, a bitmap index is accessed using a search key. If the key is found, the bitmap entry is converted to a physical row ID used to access the base table. In an index-organized table, a bitmap index is also accessed using a search key. If the key is found, the bitmap entry is converted to a physical row ID used to access the mapping table. The access to the mapping table yields a logical row ID. This logical row ID is used to access the index-organized table using either the guess data block address (if it is valid) or the primary key. Though a bitmap index on an index-organized table does not store logical row IDs, it is still logical in nature.

The movement of rows in an index-organized table does not leave the bitmap indexes built on that index-organized table unusable. Movement of rows in the index-organized table invalidate the guess data block address in some of the mapping table’s logical row ID entries, the index-organized table can still be accessed using the primary key.

Maintaining a Mapping Table

- **Collect statistics on a mapping table by analyzing the `IOT` table.**
- **Query the `DBA_INDEXES` view to determine the percentage accuracy of the mapping table.**

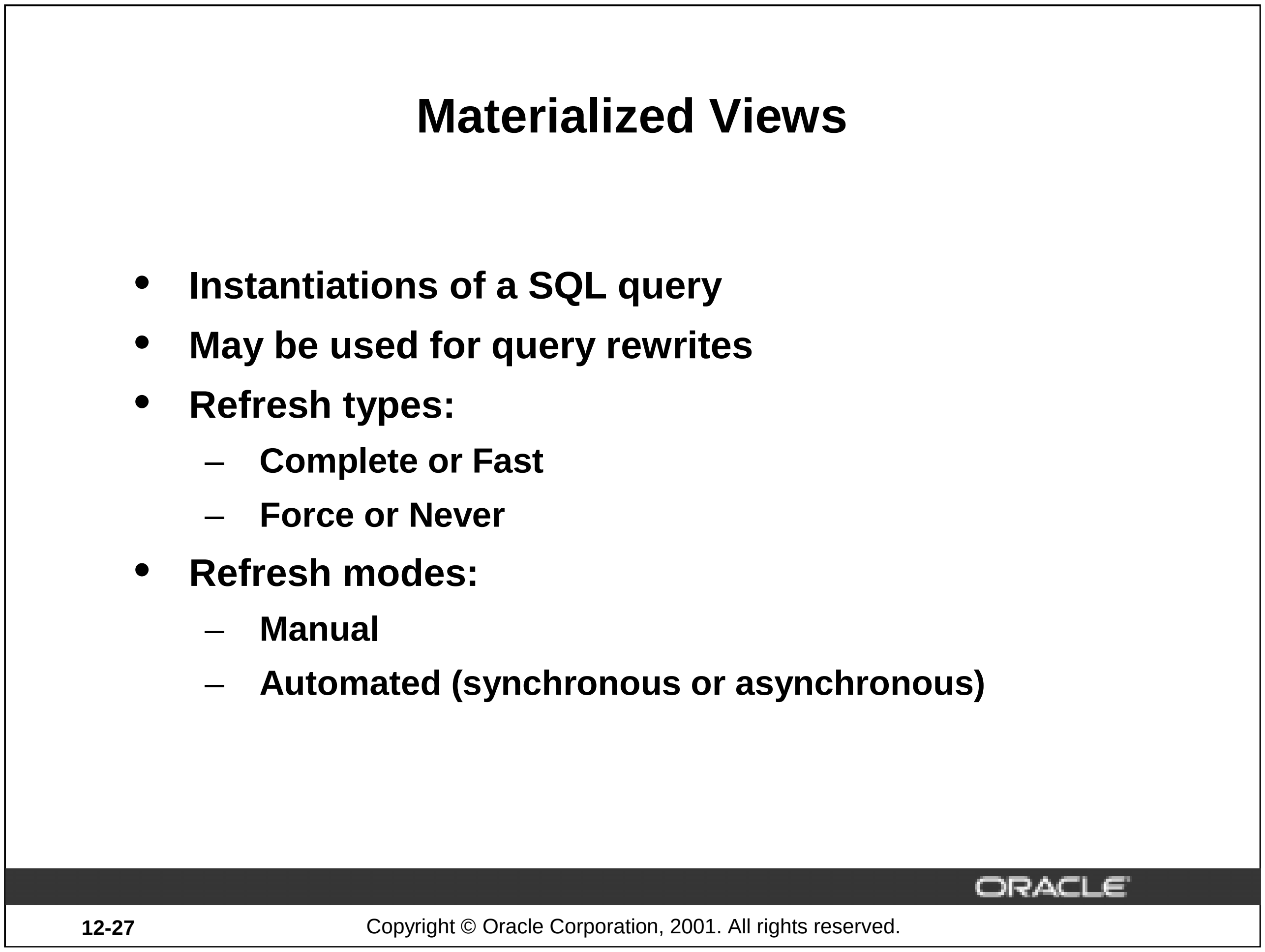
```
SQL> SELECT INDEX_NAME, PCT_DIRECT_ACCESS
  2  FROM DBA_INDEXES
  3  WHERE pct_direct_access is not null;
```

- **Rebuild the mapping table if required, using the `ALTER TABLE` command.**

Maintaining a Mapping Table

Examine the column `PCT_DIRECT_ACCESS` column of the `DBA_INDEXES`. The value in this column is an indication of the percentage of rows with `VALID` guess for a secondary index on an index-organized table. This column is populated when the IOT table is analyzed.

To rebuild the mapping table, use the `ALTER TABLE` command specifying the clause `MAPPING TABLE UPDATE BLOCK REFERENCES`.



Materialized Views

- Instantiations of a SQL query
- May be used for query rewrites
- Refresh types:
  - Complete or Fast
  - Force or Never
- Refresh modes:
  - Manual
  - Automated (synchronous or asynchronous)

12-27

Materialized Views

A materialized view stores both the definition of a view and the rows resulting from the execution of the view. Like a view, it uses a query as the basis, but the query is executed at the time the view is created and the results are stored in a table. You can define the table with the same storage parameters as any other table and place it in the tablespace of your choice. You can also index and partition the materialized view table, like other tables, to improve the performance of queries executed against them.

When a query can be satisfied with data in a materialized view, the server transforms the query to reference the view rather than the base tables. By using a materialized view, expensive operations such as joins and aggregations do not need to be re-executed by rewriting the query against the materialized view.

Creating Materialized Views

```
SQL> CREATE MATERIALIZED VIEW
  2> depart_sal_sum as
  3> select d.department_name, sum(e.salary)
  4> from departments d, employees e
  5> where d.department_id = e.department_id
  6> group by d.department_name;
```

Refreshing Materialized Views

Materialized views use the same internal mechanism as snapshots, and support several refreshing techniques.

A complete refresh of a materialized view involves truncating existing data and reinserting all the data based on the detail tables, by re-executing the query definition from the `CREATE` command.

Fast refreshes only apply the changes made since the last refresh. Two types of fast refresh are available:

- Fast refresh using materialized view logs: In this case, all changes to the base tables are captured in a log and then applied to the materialized view.
- Fast refresh using row ID range: A materialized view can be refreshed using fast refresh after direct path loads, based on the row IDs of the new rows. Direct loader logs are required for this refresh type.

A view defined with a refresh type of Force refreshes with the fast mechanism if that is possible, or else uses a complete refresh. Force is the default refresh type.

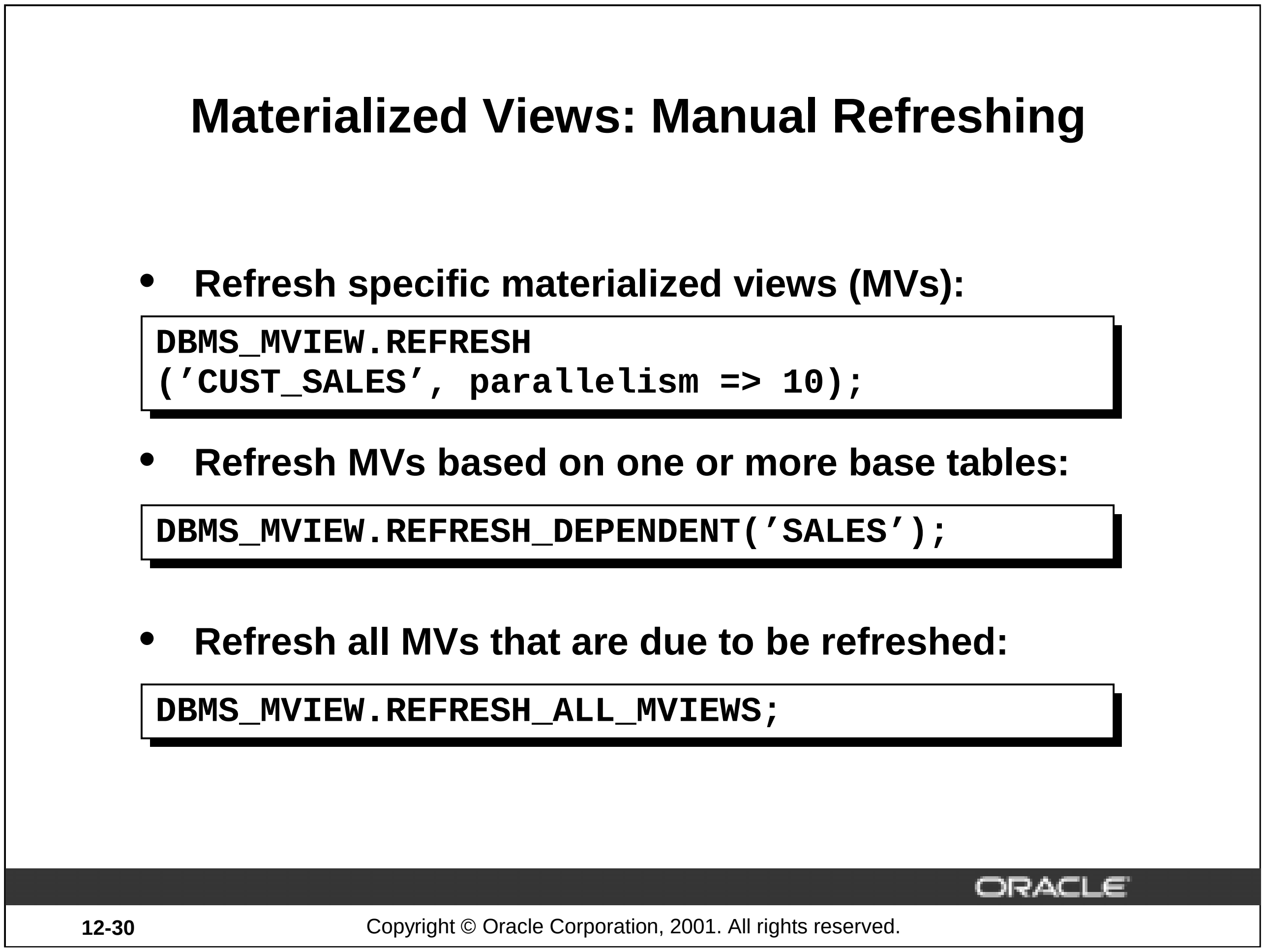
The Never option suppresses all refreshes of the materialized view.

Refresh Modes

Manual refreshes are performed using the `DBMS_MVIEW` package. The `DBMS_MVIEW` package provides a number of procedures and functions to manage materialized views, including the `REFRESH`, `REFRESH_DEPENDENT`, and `REFRESH_ALL_MVIEWS` procedures.

Automatic refreshing can be performed:

- On commit: When the `ONCOMMIT` option is specified for a materialized view, that view is updated whenever changes to one of the base tables are committed. The update to the materialized view occurs asynchronously to when the transaction is committed, and therefore does not degrade the user's perceived performance.
- At a specified time: Refreshes of a materialized view can be scheduled to occur at a specified time. For example, a view can be refreshed every Monday at 9:00 a.m. by using the `START WITH` and `NEXT` clauses. In order for such refreshes to occur, the instance must initiate job processes with the `JOB_QUEUE_PROCESSES` parameter set to a value greater than 0.



Materialized Views: Manual Refreshing

- Refresh specific materialized views (MVs):

```
DBMS_MVIEW.REFRESH
('CUST_SALES', parallelism => 10);
```

- Refresh MVs based on one or more base tables:

```
DBMS_MVIEW.REFRESH_DEPENDENT('SALES');
```

- Refresh all MVs that are due to be refreshed:

```
DBMS_MVIEW.REFRESH_ALL_MVIEWS;
```

Materialized Views: Manual Refresh

The following is a list of possible refresh scenarios for materialized views:

- Refresh specific materialized views by using the `REFRESH` procedure
- Refresh all materialized views that depend on a given set of base tables by using the `REFRESH_DEPENDENT` procedure
- Refresh all materialized views that have not been refreshed since the last bulk load to one or more detail tables by using the `REFRESH_ALL_MVIEWS` procedure

The procedures in the package use a number of parameters to specify:

- Refresh method
- Whether to proceed if an error is encountered
- Whether to use a single transaction (consistent refresh)
- Which rollback segment to use

Server job queues are used to run the refresh job. Therefore the appropriate initialization parameters, `JOB_QUEUE_PROCESSES` and `JOB_QUEUE_INTERVAL`, must be set to enable job queue refresh processes.

Query Rewrites

- **To use MVs instead of the base tables, a query must be rewritten.**
- **Query rewrites are transparent and do not require any special privileges on the MV.**
- **MVs can be enabled or disabled for query rewrites.**

12-31

Query Rewrites

Accessing a materialized view may be significantly faster than accessing the underlying base tables, so the optimizer rewrites a query to access the view when the query allows it. The query rewrite activity is transparent to applications. In this respect, their use is similar to the use of an index.

Users do not need explicit privileges on materialized views to use them. Queries executed by any user with privileges on the underlying tables can be rewritten to access the materialized view.

A materialized view can be enabled or disabled. A materialized view that is enabled is available for query rewrites.

Query Rewrites

- **The `QUERY_REWRITE_ENABLED` initialization parameter must be set to TRUE.**
- **The QUERY REWRITE privilege allows users to enable materialized views.**
- **The Summary Advisor of the `DBMS_OLAP` package has options to use materialized views.**

Query Rewrites

The rewrite of the query is performed by the optimizer. The rewrites are transparent to the application.

The ability to perform rewrites must be enabled either at the session level or at the instance level by using the `QUERY_REWRITE_ENABLED` parameter.

In order to enable or disable individual materialized views for query rewrites, the user must have the GLOBAL QUERY REWRITE or the QUERY REWRITE system privilege. Both versions of the privilege allow users to enable materialized views in their own schema. The GLOBAL version allows users to enable any materialized views they own, whereas the simple QUERY REWRITE privilege requires that the base tables as well as the views be in the user’s schema.

Overview of the Summary Advisor in the `DBMS_OLAP` Package

Materialized views provide high performance for complex, data-intensive queries. The summary advisor helps you achieve this performance benefit by choosing the proper set of materialized views for a given workload. In general, as the number of materialized views and space allocated to materialized views is increased, query performance improves. But the additional materialized views have some cost: they consume additional storage space and must be refreshed, which increases maintenance time. The summary advisor considers these costs and makes the most cost-effective trade-offs when recommending the creation of new materialized views and evaluating the performance of existing materialized views.

Materialized Views and Query Rewrites: Example

```
SQL> create MATERIALIZED VIEW sales_summary
  2     tablespace data
  3     parallel (degree 4)
  4     BUILD IMMEDIATE REFRESH FAST
  5     ENABLE QUERY REWRITE
  6   AS
  7     select p.prod_name, sum(s.quantity_sold),
  8     sum(s.amount_sold)
  9     from   sales s, products p
 10     where s.prod_id = p.prod_id
 11     group by p.prod_name;
```

Creating a Materialized View: Syntax

The syntax is similar to the `CREATE SNAPSHOT` command. There are some additional options. In the example, the `BUILD IMMEDIATE` option is chosen to cause the materialized view to be populated when the `CREATE` command is executed. This is the default behavior. You could choose the `BUILD DEFERRED` option, which creates the structure but does not populate it until the first refresh occurs.

One other option, `ON PREBUILT TABLE`, is used when you want a pre-Oracle8*i* table to be the source of a materialized view.

The `ENABLE/DISABLE QUERY REWRITE` clause determines whether query rewrites are automatically enabled for the materialized view.

Materialized Views and Query Rewrites: Example

```
SQL> select p.prod_name, sum(s.quantity_sold),
  2    sum(s.amount_sold)
  3    from   sales s, products p
  4    where s.prod_id = p.prod_id
  5    group by p.prod_name;
```

```
OPERATION             NAME
--------------------- -----------------
SELECT STATEMENT
  TABLE ACCESS FULL    SALES_SUMMARY
```

Materialized Views and Query Rewrites

The execution plan for the query—which is formatted output from a `PLAN_TABLE` table—shows that the query did not run against the two base tables, but simply scanned the materialized view table. This example illustrates the power of materialized views and query rewrites. The table comprising the materialized view substitutes completely for the base tables, and all the operations on them, named in the query.

Enabling and Controlling Query Rewrites

- **Initialization parameters:**
  - **`OPTIMIZER_MODE`**
  - **`QUERY_REWRITE_ENABLED`**
  - **`QUERY_REWRITE_INTEGRITY`**
- **Dynamic and session-level parameters:**
  - **`QUERY_REWRITE_ENABLED`**
  - **`QUERY_REWRITE_INTEGRITY`**
- **New hints: `REWRITE` and `NOREWRITE`**
- **Dimensions**

Query Rewrite Parameters

The following parameters control query rewrites:

`OPTIMIZER_MODE`: Query rewrites are only available under cost-based optimization; if rule-based optimization is used, query rewrites do not occur. This parameter can be changed using an ALTER SESSION command.

`QUERY_REWRITE_ENABLED`: This parameter can be set to FALSE to suppress query rewrites even while using the cost-based optimizer. This parameter can be altered dynamically for the whole instance as well as for individual sessions.

`QUERY_REWRITE_INTEGRITY`: This parameter can also be reset dynamically for the instance and for an individual session. It accepts the following values:

- ENFORCED (the default) enables query rewrites only if the server can guarantee consistency. Only up-to-date materialized views and enabled validated constraints are used for query rewrites.
- TRUSTED allows query rewrites based on declared, but not necessarily enforced, relationships. All updated materialized views and constraints with RELY flag are used for query rewrites.
- STALE_TOLERATED allows query rewrites to use materialized views that have not been refreshed since the last declared DML operation and relationships.

Query Rewrite: Privileges and Hints

Query rewrites are treated similarly to execution plan paths. Therefore, there are no object privileges associated with them. Users who have access to the detail tables implicitly benefit from summary rewrites.

A hint, `REWRITE`, can be used to restrict the materialized views that are considered for query rewrites. Another hint, `NOREWRITE`, is available to suppress rewrites for a query block.

Dimensions

Dimensions are data dictionary structures that define hierarchies based on columns in existing database tables. Although they are optional, they are highly recommended because they:

- Enable additional rewrite possibilities without the use of constraints (Implementation of constraints may not be desirable in a data warehouse for performance reasons.)
- Help document dimensions and hierarchies explicitly
- Can be used by OLAP tools

For more information about dimensions, see the *Oracle9i Performance Guide and Reference* manual.

Disabling Query Rewrites: Example

```
SQL> select /*+ NOREWRITE */
  2    p.prod_name, sum(s.quantity_sold),
  3     sum(s.amount_sold)
  4     from   sales s, products p
  5     where s.prod_id = p.prod_id
  6     group by p.prod_name;
```

```
OPERATION                     NAME
----------------------------- ----------
SELECT STATEMENT
SORT
HASH JOIN
TABLE ACCESS                   PRODUCTS
. . .
```

Disabling Query Rewrites

In this example, the `NOREWRITE` hint is used to tell the optimizer not to rewrite the query to make use of the materialized view. The statement is otherwise identical to the previous example. The execution plan derived from the query shows that the original statement is executed, including the hash join between the two tables and the sort to obtain the groups.

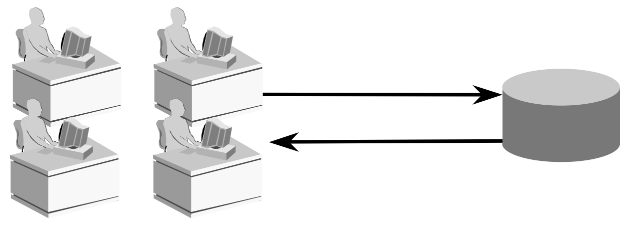
A `REWRITE/NOREWRITE` hint overrides a materialized view's definition, set in the `CREATE` or `ALTER MATERIALIZED` VIEW command with the `ENABLE QUERY REWRITE` clause.

`EXPLAIN_MV` Procedure
In this procedure, you learn what is possible with a materialized view or potential materialized view. For example, you can determine if a materialized view is fast refreshable and what types of query rewrite you can perform with a particular materialized view.

`EXPLAIN_REWRITE` Procedure
This procedure enables you to learn why a query failed to rewrite. Using the results from the procedure, you can take the appropriate action needed to make a query rewrite if at all possible. The query specified in the `EXPLAIN_REWRITE` statement is never actually executed.

OLTP Systems

- **High-throughput, insert and update-intensive**
- **Large, continuously growing data volume**
- **Concurrent access by many users**
- **Tuning goals:**
  - **Availability**
  - **Speed**
  - **Concurrency**
  - **Recoverability**



Typical OLTP Applications

- Airline reservation systems
- Large order-entry applications
- Banking applications

Requirements

- High availability (7 day/24 hour)
- High speed
- High concurrency
- Reduced time recovery

OLTP Requirements

- **Explicit extent allocation**
- **Indexes:**
  - **Not too many (B-tree better than bitmap)**
  - **Reverse key for sequence columns**
  - **Rebuilt regularly**
- **Clusters for tables in join queries:**
  - **Index clusters for growing tables**
  - **Hash clusters for stable tables**
- **Materialized views**
- **Index-organized tables**

ORACLE

12-39

OLTP Requirements

Space Allocation

- Avoid the performance load of dynamic extent allocation; allocate space explicitly to tables, clusters and indexes.
- Check growth patterns regularly to find the rate at which extents are being allocated, so that you can create extents appropriately.

Indexing

- Indexing is critical to data retrieval in OLTP systems. DML statements on indexed tables need index maintenance, and this is a significant performance overhead. Your indexing strategy must be closely geared to the real needs of the application.
- Indexing a foreign key helps child data to be modified without locking the parent data.
- B-tree indexing is better than bitmap indexing, because of locking issues affecting DML operations: when a B-tree index entry is locked, a single row is locked, whereas when a bitmap index entry is locked, a whole range of rows are locked.
- Reverse key indexes avoid frequent B-tree block splits for sequence columns.
- You should rebuild indexes regularly.

OLTP Requirements (continued)

Hash Clustering

- Using hash clusters should speed up access on equality queries. But you should not choose this data structure for a table that is:
  - Heavily inserted into, because of the use of space and the number of blocks that need to be visited
  - Heavily updated using larger column values, because of migration probability
- If a table is growing, there are likely to be many collisions on hash keys and this leads to storing rows in overflow blocks. To avoid this situation, correctly estimate the HASHKEYS value. With a large number of hash keys for a given number of rows, the likelihood of a collision decreases.

OLTP Application Issues

- **Use declarative constraints instead of application code.**
- **Make sure that code is shared.**
- **Use bind variables rather than literals for optimally shared SQL.**
- **Use the `CURSOR_SHARING` parameter.**

Integrity Constraints

If there is a choice between keeping application logic in procedural code or using declarative constraints, bear in mind that constraints are always less expensive to process. Referential integrity and CHECK constraints are the main types to consider here.

Shared Code

Otherwise, you should make certain that code is shared by stored procedural objects, such as packages, procedures, and functions.

Bind Variables

You want to keep the overhead of parsing to a minimum. Try to ensure that the application code uses bind variables rather than literals.

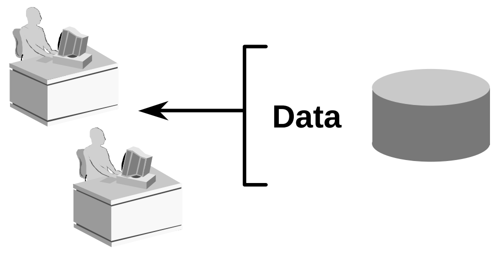
The `CURSOR_SHARING` Parameter

Using the `CURSOR_SHARING` parameter can help users share parsed code. The following vales are allowed:

- EXACT (the default) shares cursors only for exact SQL statements.
- SIMILAR Oracle9i will share cursors if changing literals to bind variables will generate the same execution path as would have been used if the parameter was set to EXACT.
- FORCE shares cursors by changing literals to bind variables, even if the execution path changes.

Decision Support Systems (Data Warehouses)

- Queries on large amounts of data
- Heavy use of full table scans
- Tuning goals:
  – Fast response time
  – Accuracy
- The Parallel Query feature is designed for data warehouse environments.



Characteristics of Decision Systems

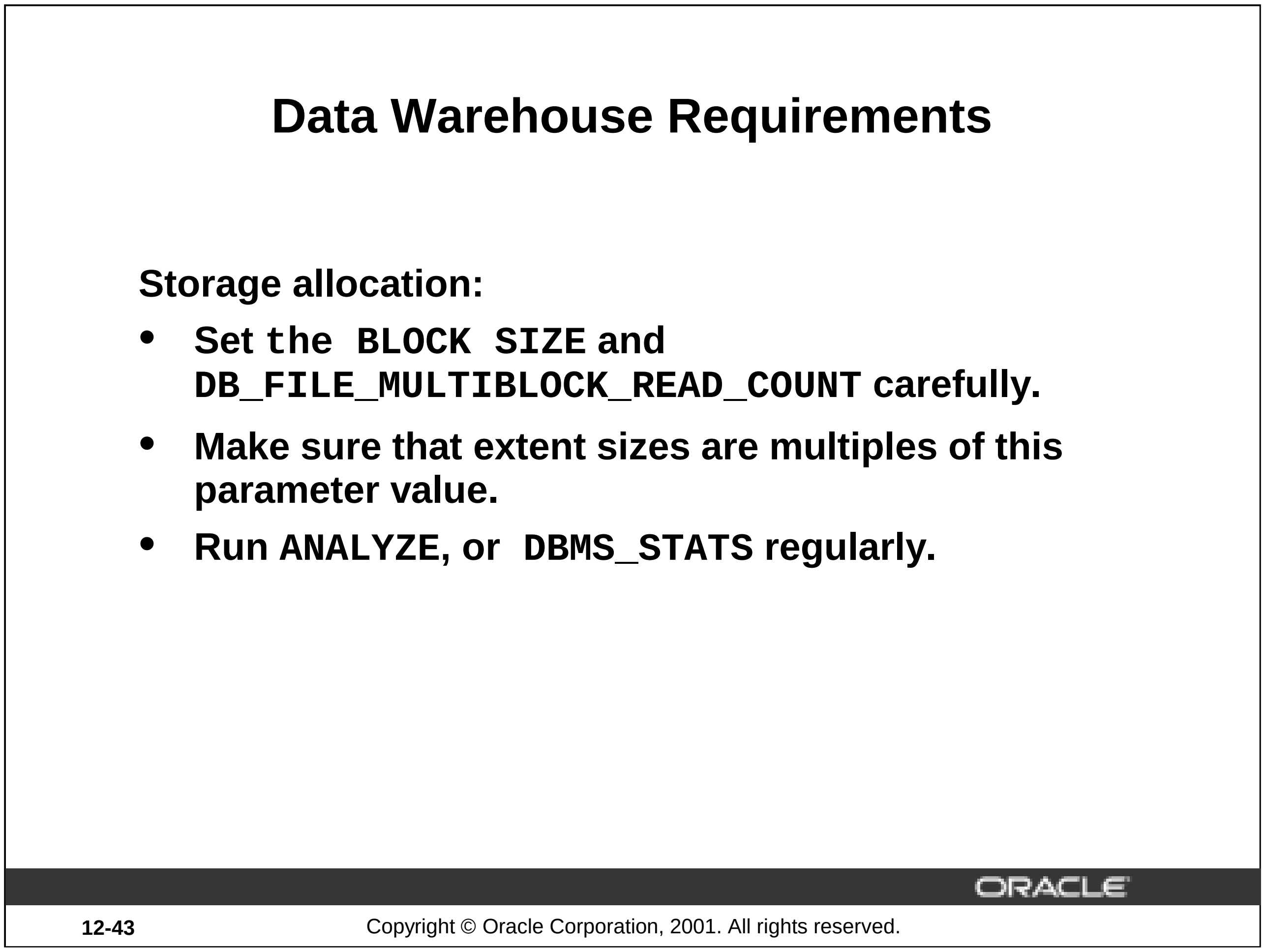
Decision support applications distill large amounts of information into understandable reports. They gather data from OLTP systems and use summaries and aggregates in retrieving it. They make heavy use of full table scans as an access method.

Decision-makers in an organization use these data to determine what strategies the organization should take.

Example: A marketing tool determines the buying patterns of consumers based on information gathered from demographic data, to determine which items sell best in which locations.

Requirements

- Fast response time: When you design a data warehouse, make sure that queries on large amounts of data can be performed within a reasonable time frame.
- Accuracy



Data Warehouse Requirements

Storage allocation:

- **Set `the BLOCK SIZE` and `DB_FILE_MULTIBLOCK_READ_COUNT` carefully.**
- **Make sure that extent sizes are multiples of this parameter value.**
- **Run `ANALYZE`, or `DBMS_STATS` regularly.**

`BLOCK SIZE` and `DB_FILE_MULTIBLOCK_READ_COUNT`

Because data warehouse applications typically perform many table scans, consider a higher value for the `block size` parameter. Even if this means re-creating a large database, it almost certainly worthwhile, because a larger block size facilitates read-intensive operations, which are characteristic of data warehouse applications. With Oracle9*i* another option is available, creating a new tablespace with the required block size. This is possible because Oracle9*i* allows tablespaces to have different block sizes.

Pay particular attention to the setting of the `DB_FILE_MULTIBLOCK_READ _COUNT` parameter. During full table scans and fast full index scans this parameter determines how many database blocks are read into the buffer cache with a single operating system read call. A larger value decreases estimated table scan costs and favors table scans over index searches.

Data Warehouse Requirements

- **Evaluate the need for indexes:**
  - **Use bitmap indexes when possible.**
  - **Use index-organized tables for (range) retrieval by primary keys.**
  - **Generate histograms for indexed columns that are not distributed uniformly.**
- **Clustering: Consider hash clusters for performance access.**

Indexing

You should minimize the space and performance overhead of index maintenance. Because most queries use full table scans, you can:

- Dispense with indexes altogether
- Maintain them only for a few tables that are accessed selectively
- Regularly generate histograms for tables with nonuniformly distributed data
- Choose bitmap indexes for queries on columns with few distinct values; they offer much faster retrieval access
- Use index-organized tables for faster, key-based access to table data for queries involving exact matches or range searches and complete row data retrieval

Clustering

Consider using both types of clusters, particularly hash clusters. Do not cluster tables that grow regularly during bulk loads.

Data Warehouse Application Issues

- **Parsing time is less important.**
- **The execution plan must be optimal:**
  - **Use the Parallel Query feature.**
  - **Tune carefully, using hints if appropriate.**
  - **Test on realistic amounts of data.**
  - **Consider using PL/SQL functions to code logic into queries.**
- **Bind variables are problematic.**

Parsing Time

The time taken to parse `SELECT` statements is likely to be a very small proportion of the time taken to execute the query. Tuning the library cache is much less of an issue for data warehouse than for OLTP systems.

Your priority is an optimal access path in the execution plan; small variations can cost minutes or hours. Developers must:

- Use *parallelized queries* which enable multiple processes to work together simultaneously to process a single SQL statement

  Symmetric multiprocessors (SMP), clustered, or massively parallel processing (MPP) configurations gain the largest performance benefits, because the operation can be effectively split among many CPUs on a single system.
- Use the `EXPLAIN PLAN` command to tune the SQL statements, and *hints* to control access paths

If your application logic uses bind variables, you lose the benefit of this feature: the optimizer makes a blanket assumption about the selectivity of the column, whereas with literals, the cost-based optimizer uses histograms. Be careful when setting the `CURSOR_SHARING` because it could force a change from literal values, to system generated bind variables.

Hybrid Systems

OLTP	Data Warehouse
Performs index searches	More full table scans
Uses B-tree indexes	Uses bitmap indexes
Uses reverse key indexes	Uses index-organized tables
`CURSOR_SHARING` set to SIMILAR can assist performance	`CURSOR_SHARING` should be left on EXACT
Should not use Parallel Query	Employs Parallel Query for large operations
PCTFREE according to expected update activity	PCTFREE can be set to 0
Shared code and bind variables	Literal variables and hints
Uses `ANALYZE` indexes	Generates histograms

Hybrid System Issues

- OLTP needs more indexes than Data Warehouse does. They do not necessarily use the same type of indexes, because of DML/OLTP and queries/data warehouse issues.
- OLTP should not use parallel query, whereas Data Warehouse needs it.
- Data Warehouse requires tightly packed data blocks for optimal full table scans. Individual rows are generally not updated in a data warehouse environment. Therefore, `PCTFREE` should be set to 0 for data warehouse databases, whereas a `PCTFREE` of 0 for OLTP will lead to chaining, because rows are typically more dynamic.
- The cost-based optimizer takes histogram statistics into account when queries use literals and not bind variables. While OLTP must use bind variables, Data Warehouse does not. This means that histogram statistics collection is a loss of time and consumes resources for OLTP transactions.

Summary

In this lesson, you should have learned how to describe the following:

- The role of the DBA in tuning applications
- Different storage structures, and why one storage structure might be preferred over another
- The different types of indexes
- Index-organized tables
- Materialized views and the use of query rewrites
- Requirements for OLTP, DSS, and hybrid systems

Practice 12

In this practice you will make use of the AUTOTRACE feature, and create the table plan_table. These are covered in detail in the chapter titled “SQL Statement Tuning”

1. Connect as hr/hr and create an IOT called ‘New_employees’ in the ‘HR’ schema. Give the table the same columns as the hr.employees table. Make the column employee_id the primary key, and name the primary key index new_employees_employee_id_pk
2. Confirm the creation of the table by querying the user_tables, and the user_indexes views
   – The IOT is a table, and so will be found in the user_tables view.
   – The IOT is an index, and so will be found in the user_indexes view.
3. Populate the new_employees table with the rows from the hr.employees table.
4. Create a secondary B-Tree index on the column last_name of the new_employees table. Place the index in the INDX tablespace. Name the index last_name_new_employees_idx. Use the Analyze Index command to collect the statistics for the secondary index.
5. Confirm the creation of the index by using the user_indexes view in the data dictionary. Query the index_name, index_type, blevel, and leaf_blocks.

   Note: If the values for blevel and leaf blocks are null then there were no statistics collected. Confirm that the value of index_type is normal.
6. Create a reverse key index on the department_id of the employees_hist table. Place the index in the INDX tablespace. Name the index emp_hist_dept_id_idx.
7. Confirm the creation of the index, and that it is a reverse key index, by querying the user_indexes view in the data dictionary. Query the index_name, index_type, blevel, and leaf_blocks.

   Note: This time the values of blevel and leaf blocks should be null as you did not collect statistics for this index while creating it. Also the value for index type should now be normal/reverse.
8. Create a bitmap index on the job_id column of the employees_hist table. Place the index in the INDX tablespace. Name the index bitmap_emp_hist_idx.
9. Confirm the creation of the index, and that it is a bitmapped index , by querying the user_indexes view in the data dictionary. Query the index_name, index_type, blevel, and leaf_blocks.
10. Connect as sh/sh, and confirm that the PLAN_TABLE exists. If the table does exist then truncate it, otherwise create the PLAN_TABLE using $ORACLE_HOME/rdbms/admin/utlxplan.sql.
11. Create a Materialized View cust_sales having two columns, cust_last_name and the total sales for that customer. This will mean joining the sales and customers tables using cust_id, and grouping the results by cust_last_name. Make sure that query rewrite is enabled on the view.

Practice 12 (continued)

12. Confirm the creation of the Materialized View cust_sales by querying the data dictionary view USER_MVIEWS, selecting the columns mview_name, rewrite_enabled and query.

    Note: Rewrite_enabled must be set to Y in order for the practice on query rewrite to work.

13. Set AUTOTRACE to trace only explain, to generate the explain plan for the query

14. Set the query_rewrite_enabled parameter to true for the session and run the same query, $HOME/STUDENT/LABS/lab12_13.sql, as in the previous practice. Note the change in the explain plan due to the query rewrite. Set AUTOTRACE off and disable query rewrite after the script has completed running.

13

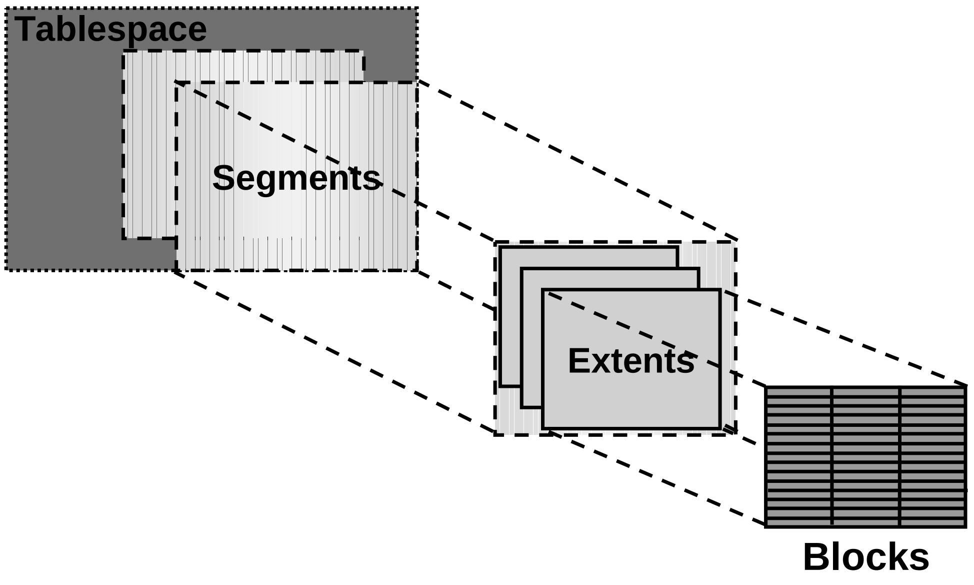
Using Oracle Blocks Efficiently

Objectives

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

- Describe the correct usage of extents and Oracle blocks
- Explain space usage and the high-water mark
- Determine the high-water mark
- Describe the use of Oracle Block parameters
- Recover space from sparsely populated segments
- Describe and detect chaining and migration of Oracle blocks
- Perform index reorganization
- Monitor indexes to determine usage

Database Storage Hierarchy



Space Management

The efficient management of space in the database is important to its performance. This section of the lesson examines how to manage extents and blocks in the database.

Blocks

In an Oracle database, the block is the smallest unit of data file I/O and the smallest unit of space that can be allocated. An Oracle block consists of one or more contiguous operating system blocks.

Extents

An extent is a logical unit of database storage space allocation made up of a number of contiguous data blocks. One or more extents make up a segment. When the existing space in a segment is completely used, the Oracle server allocates a new extent for the segment.

Segments

A segment is a set of extents that contains all the data for a specific logical storage structure within a tablespace. For example, for each table, the Oracle server allocates one or more extents to form data segments for that table. For indexes, the Oracle server allocates one or more extents to form its index segment.

Allocation of Extents

To avoid the disadvantages of dynamic extent allocation:

- **Create locally managed tablespaces.**
- **Size the segments appropriately.**
- **Monitor segments ready to extend.**

Allocation of Extents

When database operations cause the data to grow and exceed the space allocated, the Oracle server extends the segment. Dynamic extension, or extending the segment when executing an `INSERT` or `UPDATE` statement, reduces performance because the server executes several recursive SQL statements to find free space and add the extent to the data dictionary. However, this is not the case for locally managed tablespaces that avoid recursive space management operations.

Avoiding Dynamic Allocation

To display segments with less than 10% free blocks:

```
SQL> SELECT  owner, table_name, blocks, empty_blocks
  2    FROM  dba_tables
  3    WHERE empty_blocks / (blocks+empty_blocks) < .1;
OWNER  TABLE_NAME     BLOCKS EMPTY_BLOCKS
------ ---------- ---------- ------------
HR     EMPLOYEES        1450           50
HR     COUNTRIES         460           40
```

To avoid dynamic allocation:

```
SQL> ALTER TABLE hr.employees ALLOCATE EXTENT;
Table altered.
```

ORACLE

Avoid Dynamic Extension

- Size the segment appropriately, by:
  - Determining the maximum size of your object
  - Choosing storage parameters that allocate extents large enough to accommodate all of your data when you create the object

  When determining the segment size, the DBA should allow for the growth of data. For example, allocate enough space for the current data and for any data that will be inserted into the segment over the next year. For formulas to predict how much space to allow for a table, see the *Oracle9i Server Administrator's Guide*.
- Monitor the database for segments that are about to dynamically extend, and extend them with an `ALTER TABLE/INDEX/CLUSTER` command.

Locally Managed Extents

- **Create a locally-managed tablespace:**

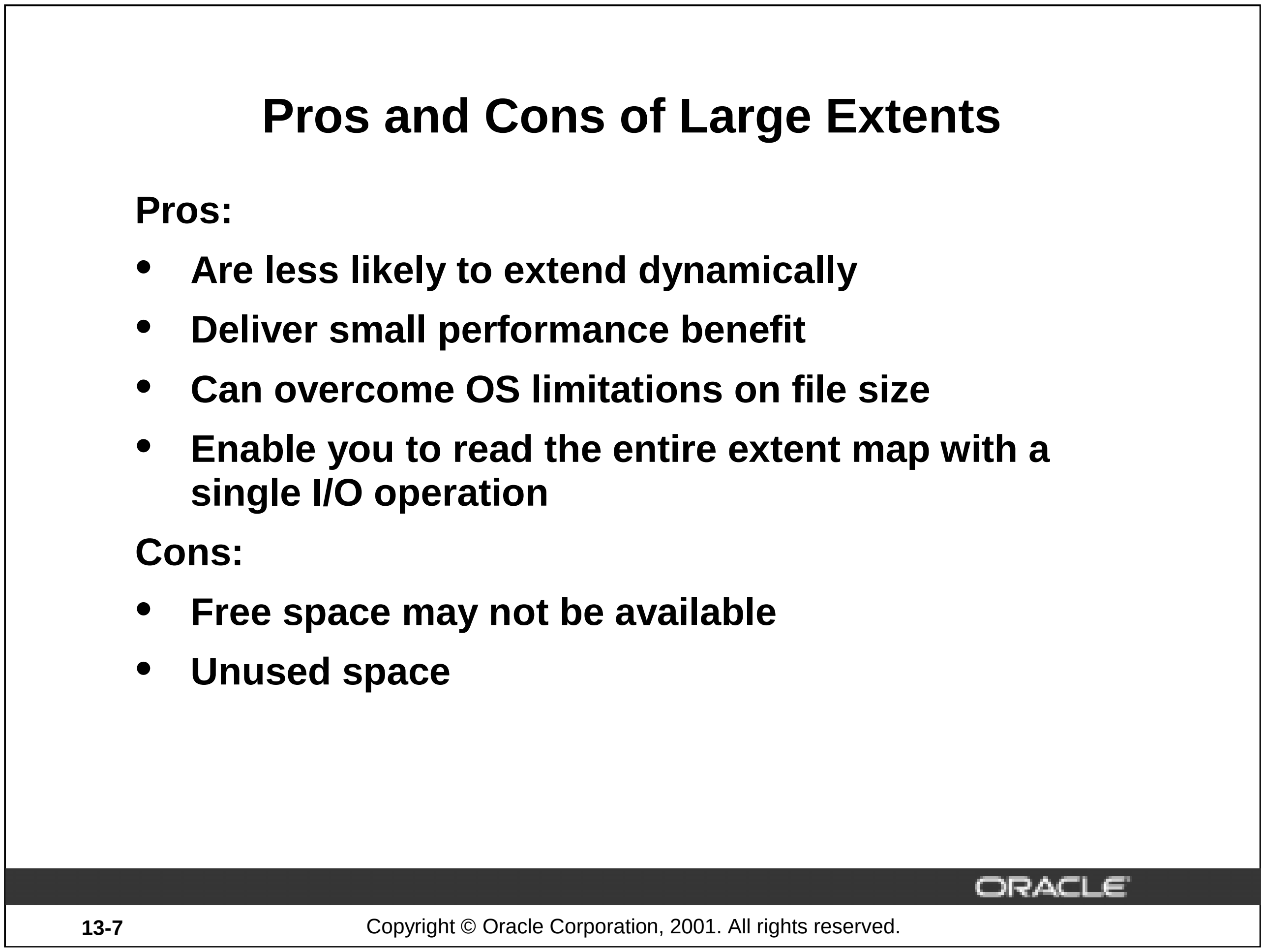
```
CREATE TABLESPACE user_data_1
DATAFILE '/oracle9i/oradata/db1/lm_1.dbf'
SIZE 100M
EXTENT MANAGEMENT LOCAL
UNIFORM SIZE 2M;
```

- **With Oracle9*i*, the default for extent management is local.**

Locally Managed Extents

Create locally managed tablespaces for the objects that extend continuously.

A locally managed tablespace manages its own extents and 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 bitmap values change to show the new status of the blocks. These changes do not generate rollback information because they do not update tables in the data dictionary.



Pros and Cons of Large Extents

Pros:

- **Are less likely to extend dynamically**
- **Deliver small performance benefit**
- **Can overcome OS limitations on file size**
- **Enable you to read the entire extent map with a single I/O operation**

Cons:

- **Free space may not be available**
- **Unused space**

13-7

Advantages of Large Extents

To ease space management, the DBA should create objects with appropriate size segments and extents. As a general rule, larger extents are preferred over smaller extents.

- Large extents avoid dynamic extent allocation, because segments with larger extents are less likely to need to be extended.
- Larger extents can have a small performance benefit because the Oracle server can read one large extent from disk with fewer multiblock reads than would be required to read many small extents. To avoid partial multiblock reads, set the extent size to a multiple of 5 × `DB_FILE_MULTIBLOCK_READ_COUNT`. Multiply by five because the Oracle server tries to allocate extents on five-block boundaries. By matching extent sizes to the I/O and space allocation sizes, the performance cost of having many extents in a segment is minimized. However, for a table that never has a full table scan operation, it makes no difference in terms of query performance whether the table has one extent or multiple extents.
- For very large tables, operating system limitations on file size may force the DBA to allocate the object with multiple extents.
- The performance of searches using an index is not affected by the index having one extent or multiple extents.

Advantages of Large Extents (continued)

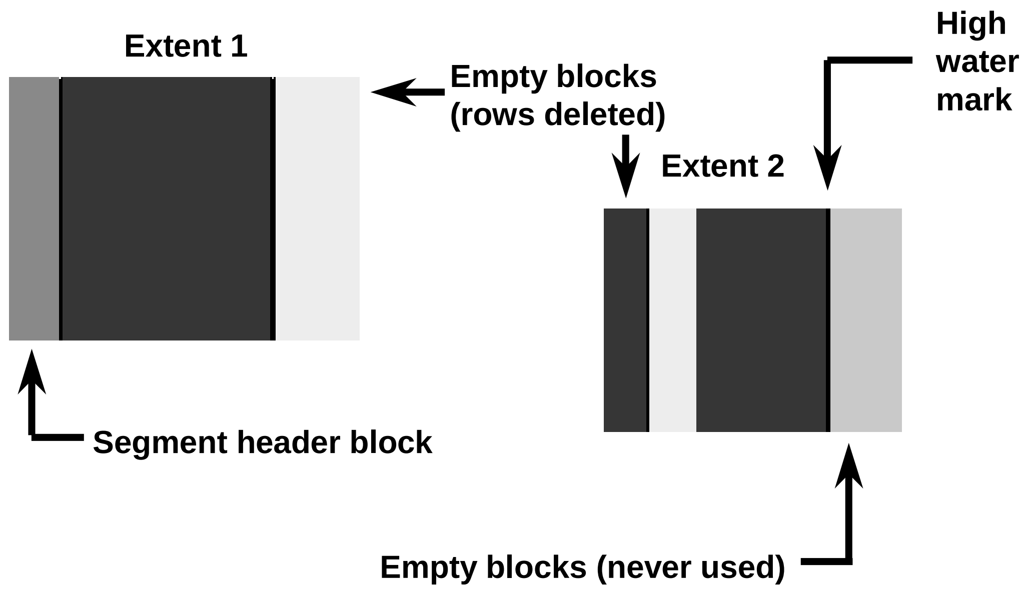
- Extent maps list all the extents for a certain segment. When `MAXEXTENTS` is set to `UNLIMITED`, these maps are in multiple blocks. For best performance, you should be able to read the extent map with a single I/O. Performance degrades if multiple I/Os are necessary for a full table scan to get the extent map. Also, a large number of extents can degrade data dictionary performance, because each extent uses space in the dictionary cache.

Disadvantages of Large Extents

- Because large extents require more contiguous blocks, the Oracle server may have difficulty finding enough contiguous space to store them.
- Because the DBA sizes the segment to allow for growth, some of the space allocated to the segment is not used initially.

To determine whether to allocate few large extents or many small extents, consider how the benefits and drawbacks of each would affect your plans for the growth and use of your tables.

The High-Water Mark



The High-Water Mark

Note the use of empty blocks to describe two different types of blocks.

Empty Blocks (rows deleted) Shown in Extent 1 and Extent 2

These empty blocks have been used by the table to store data that has now been deleted. These blocks, assigned to the free list of the table, are to be used by `INSERT` statements.

Empty Blocks (never used) Shown in Extent 2 only

Empty blocks in this region are those blocks assigned to the table, but which have not yet been used by the table, and thus are found above the high-water mark.

The High-Water Mark

The high-water mark is:

- Recorded in segment header block
- Set to the beginning of the segment on creation
- Incremented in five-block increments as rows are inserted
- Reset by the `TRUNCATE` command
- Never reset by `DELETE` statements

The High-Water Mark

Space above the high-water mark can be reclaimed at the table level by using the following command:

```
ALTER TABLE <table_name> DEALLOCATE UNUSED...
```

In a full table scan, the Oracle server reads in all blocks below the high-water mark. Empty blocks above the high-water mark may waste space, but should not degrade performance; however, underused blocks below the high-water mark may degrade performance.

Clusters

In clusters, space is allocated for all cluster keys, whether they contain data or not. The amount of space allocated depends on the value of the SIZE parameter specified when creating the cluster, and the type of cluster:

- In a hash cluster, because the number of hash keys is specified in the create cluster statement, there is space allocated below the high-water mark for every hash key.
- In an index cluster, there is space allocated for every entry into the cluster index.

Table Statistics

Populate the table statistics with the ANALYZE command and then query the values in DBA_TABLES:

```
SQL> ANALYZE TABLE hr.employees COMPUTE STATISTICS;
Table analyzed.

SQL> SELECT  num_rows, blocks, empty_blocks as empty,
  2          avg_space, chain_cnt, avg_row_len
  3  FROM    dba_tables
  4  WHERE   owner = 'HR'
  5  AND     table_name = 'EMPLOYEES';
NUM_ROWS BLOCKS EMPTY AVG_SPACE CHAIN_CNT AVG_ROW_LEN
-------- ------ ----- --------- --------- -----------
   13214    615    35      1753         0         184
```

Table Statistics

You can analyze the storage characteristics of tables, indexes, and clusters to gather statistics, which are then stored in the data dictionary. You can use these statistics to determine whether a table or index has unused space.

Query the DBA_TABLES view to see the resulting statistics:

Num_Rows	- Number of rows in the table
Blocks	- Number of blocks below the high-water mark of the table
Empty_blocks	- Number of blocks above the high-water mark of the table
Avg_space	- Average free space in bytes in the blocks below the high-water mark
Avg_row_len	- Average row length, including row overhead
Chain_cnt	- Number of chained, or migrated, rows in the table
Avg_space_freelist_blocks	- The average freespace of all blocks on a freelist
Num_freelist_blocks	- The number of blocks on the freelist

EMPTY_BLOCKS represents blocks that have not yet been used, rather than blocks that were full and are now empty.

The DBMS_SPACE Package

```
declare
     owner       VARCHAR2(30);
     table_name  VARCHAR2(30);
     seg_type    VARCHAR2(30);
     tblock      NUMBER;
...
BEGIN
  dbms_space.unused_space
    ('&owner','&table_name','TABLE',
     tblock,tbyte,ublock,ubyte,lue_fid,lue_bid,lublock);

  dbms_output.put_line(...
END;
/
```

The DBMS_SPACE Package

You can also use this supplied package to get information about space use in segments. It contains two procedures:

- UNUSED_SPACE returns information about unused space in an object (table, index, or cluster). Its specification is:

```
unused_space (segment_owner IN       varchar2,
  segment_name IN                    varchar2,
  segment_type IN                    varchar2,
  total_blocks OUT                   number,
  total_bytes OUT                    number,
  unused_blocks OUT                  number,
  unused_bytes OUT                   number,
  last_used_extent_file_id OUT       number,
  last_used_extent_block_id OUT      number,
  last_used_block OUT                number);
```

The `DBMS_SPACE` Package (continued)

- `FREE_BLOCKS` returns information about free blocks in an object (table, index, or cluster). Its specification is:

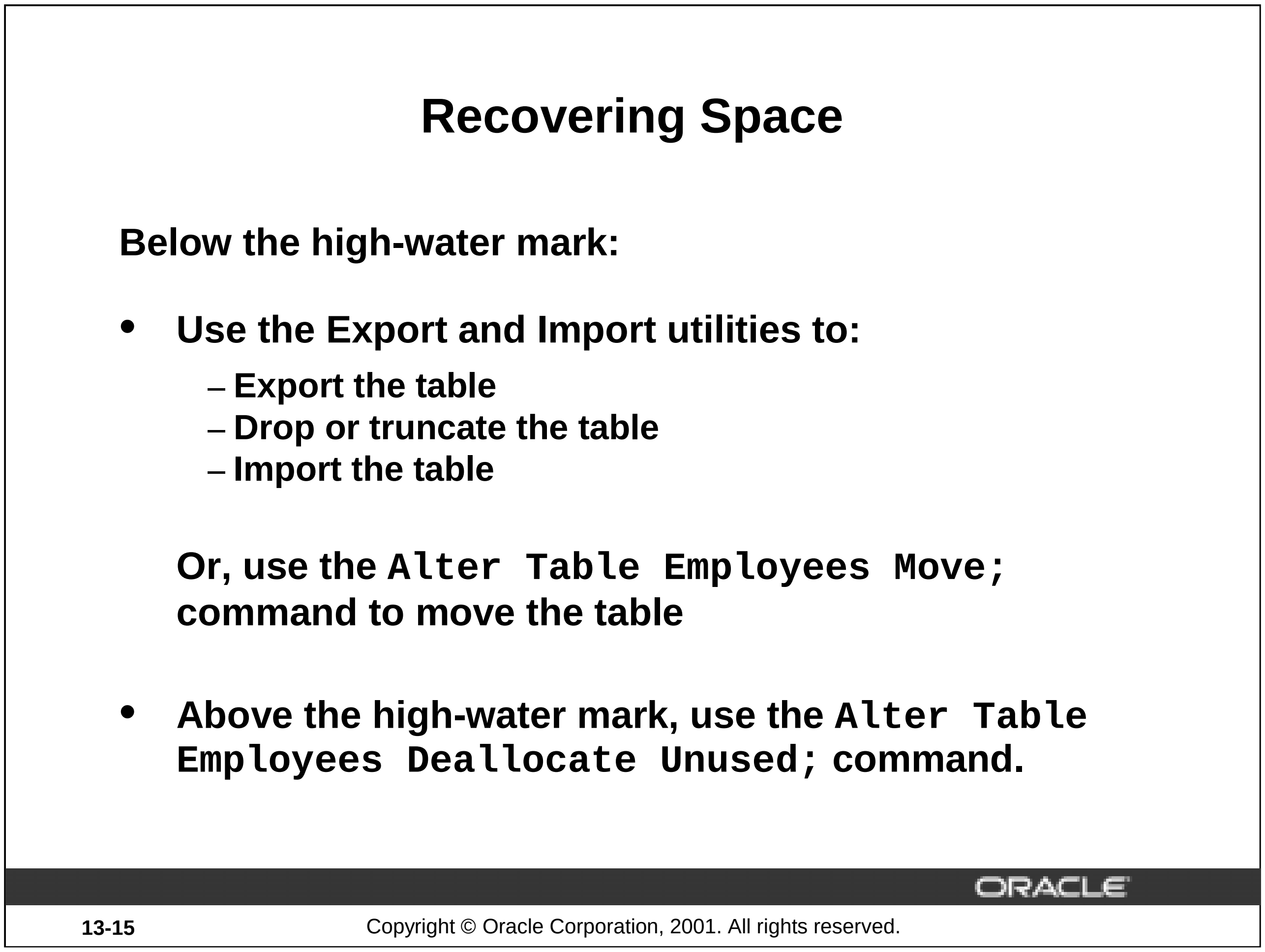
```
free_blocks (segment_owner IN      varchar2,
  segment_name IN                  varchar2,
  segment_type IN                  varchar2,
  freelist_group_id  IN            number,
  free_blks OUT                    number,
  scan_limit IN                    number DEFAULT NULL);
```

These procedures are created by and documented in the `dbmsutil.sql` script that is run by `catproc.sql`. When running this package, you must provide a value for the `FREE_LIST_GROUP_ID`. Use a value of 1, unless you are using Oracle Parallel Server.

The DBMS_SPACE Package (continued)

The following script prompts the user for the table owner and table name, executes `DBMS_SPACE.UNUSED_SPACE`, and displays the space statistics.

```
 declare
     owner varchar2(30);
     name  varchar2(30);
     seg_type          varchar2(30);
     tblock            number;
     tbyte number;
     uBlock            number;
     ubyte number;
     lue_fid           number;
     lue_bid           number;
     lublock           number;
BEGIN
 dbms_space.unused_space('&owner','&table_name','TABLE',
     tblock,tbyte,ublock,ubyte,lue_fid,lue_bid,lublock);
 dbms_output.put_line('Total blocks allocated to table    = '
     || to_char(tblock));
 dbms_output.put_line('Total bytes allocated to table     = '
     ||to_char(tbyte));
 dbms_output.put_line('Unused blocks(above HWM)           = '
     ||to_char(ublock));
 dbms_output.put_line('Unused bytes(above HWM)            = '
     ||to_char(ubyte));
 dbms_output.put_line('Last extent used file id           = '
     ||to_char(lue_fid));
 dbms_output.put_line('Last extent used begining block id = '
     ||to_char(lue_bid));
 dbms_output.put_line('Last used block in last extent     = '
     ||to_char(lublock));
END;
```



Recovering Space

Below the high-water mark:

- **Use the Export and Import utilities to:**
  - **Export the table**
  - **Drop or truncate the table**
  - **Import the table**

  Or, use the `Alter Table Employees Move;` command to move the table

- **Above the high-water mark, use the `Alter Table Employees Deallocate Unused;` command.**

13-15

Dropping or Truncating the Table

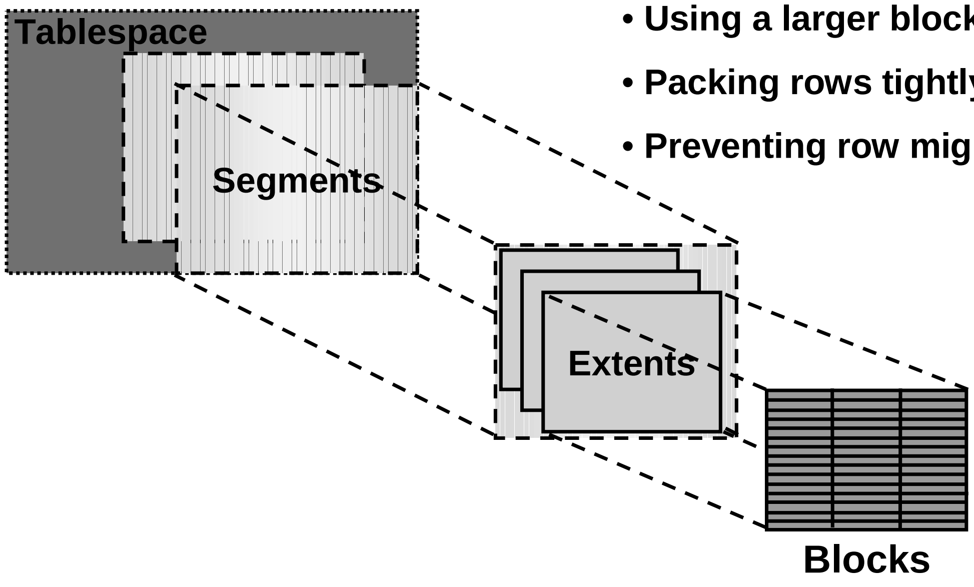
When deciding whether to drop or truncate the table, consider the following:

- Both actions have the same result: no data in the table.
- A drop removes from the data dictionary all information regarding this table. For example, the extents used by the table will be deallocated.
- Truncate has the option to keep all allocated space, by specifying reuse storage.
- If using the drop table, careful consideration must be given to using the *compress* option, because there might not be a single contiguous area large enough for the entire space allocated to the table when importing.
- If the table is stored in a dictionary managed tablespace, then the deallocation (at the drop, or truncate - if using the default setting) and allocation (at the import stage)of extents could be a major time factor, depending on the number of extents (not the size).

Move the table

After the table is moved, all indexes are marked unusable, and must be rebuilt.

Database Block Size



Minimize block visits by:

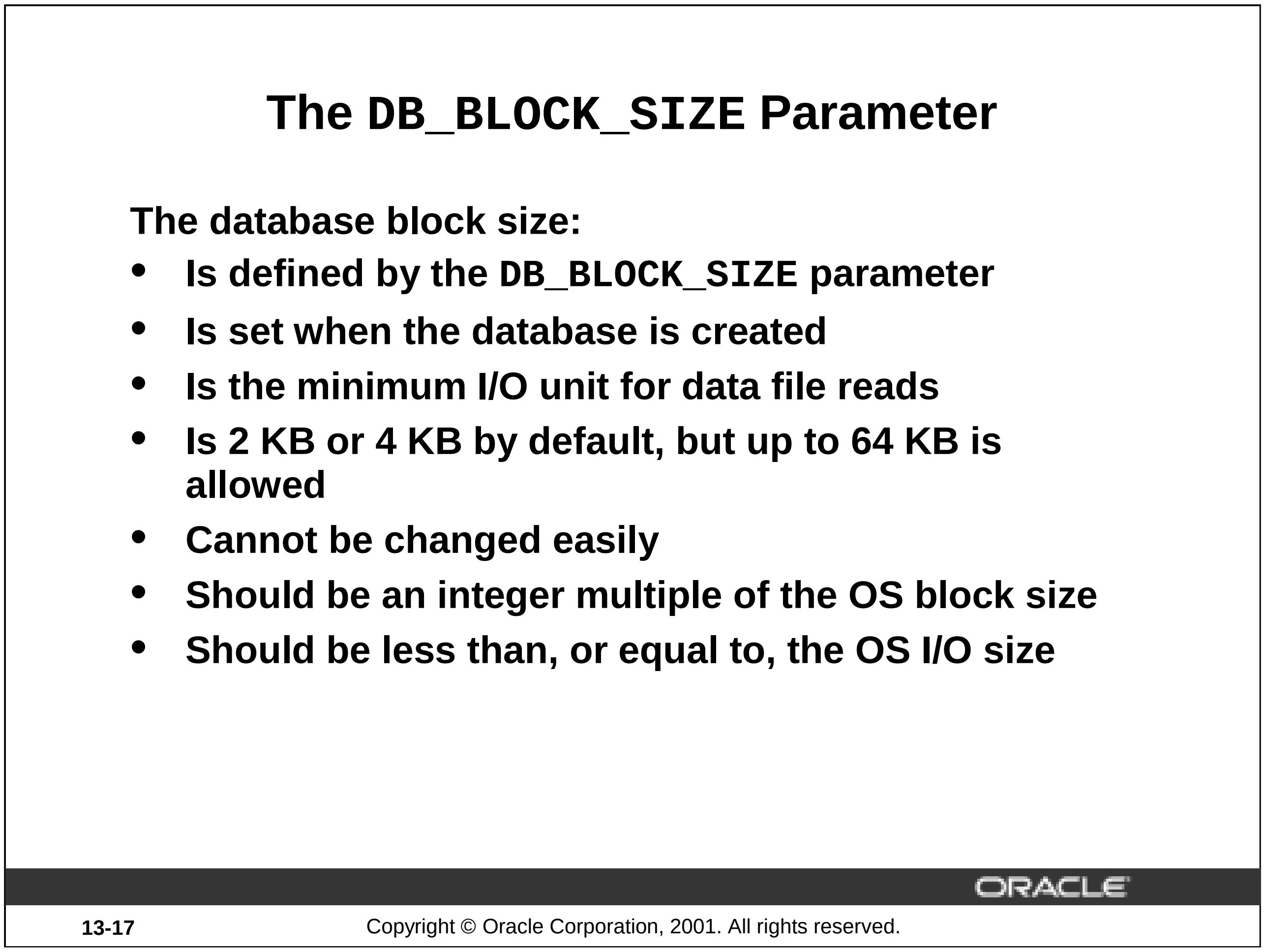
- **Using a larger block size**
- **Packing rows tightly**
- **Preventing row migration**

Minimizing the Number of Block Visits

One of the database tuning goals is to minimize the number of blocks visited. The developer contributes to this goal by tuning the application and SQL statements. The DBA reduces block visits by:

- Using a larger block size
- Packing rows as closely as possible into blocks
- Preventing row migration

Unfortunately for the DBA, the last two goals conflict: as more data is packed into a block, the likelihood of migration increases.



The DB_BLOCK_SIZE Parameter

The database block size:

- Is defined by the DB_BLOCK_SIZE parameter
- Is set when the database is created
- Is the minimum I/O unit for data file reads
- Is 2 KB or 4 KB by default, but up to 64 KB is allowed
- Cannot be changed easily
- Should be an integer multiple of the OS block size
- Should be less than, or equal to, the OS I/O size

13-17

Characteristics

- When the database is created, its block size is set to the value of DB_BLOCK_SIZE parameter.
- It is the minimum I/O unit for data file reads.
- The default block size on most Oracle platforms is either 2 or 4 KB.
- Some operating systems now allow block sizes of up to 64 KB. Check with your operating system–specific documentation, specifically the Oracle installation and configuration guides for your platform.
- The size cannot be changed without re-creating or duplicating the database. This makes it difficult to test applications with different block sizes. This restriction is partially lifted with Oracle9*i* where a database can have multiple block sizes. However, the base block size (that of the system tablespace) cannot be changed.
- The database block size should be a multiple of the operating system block size.
- If your operating system reads the next block during sequential reads, and your application performs many full table scans, then the database block size should be large, but not exceeding the operating system I/O size.

Small Block Size: Pros and Cons

Pros:

- **Reduces block contention**
- **Is good for small rows**
- **Is good for random access**

Cons:

- **Has relatively large overhead**
- **Has small number of rows per block**
- **May cause more index blocks to be read**

Small Oracle Blocks

Advantages

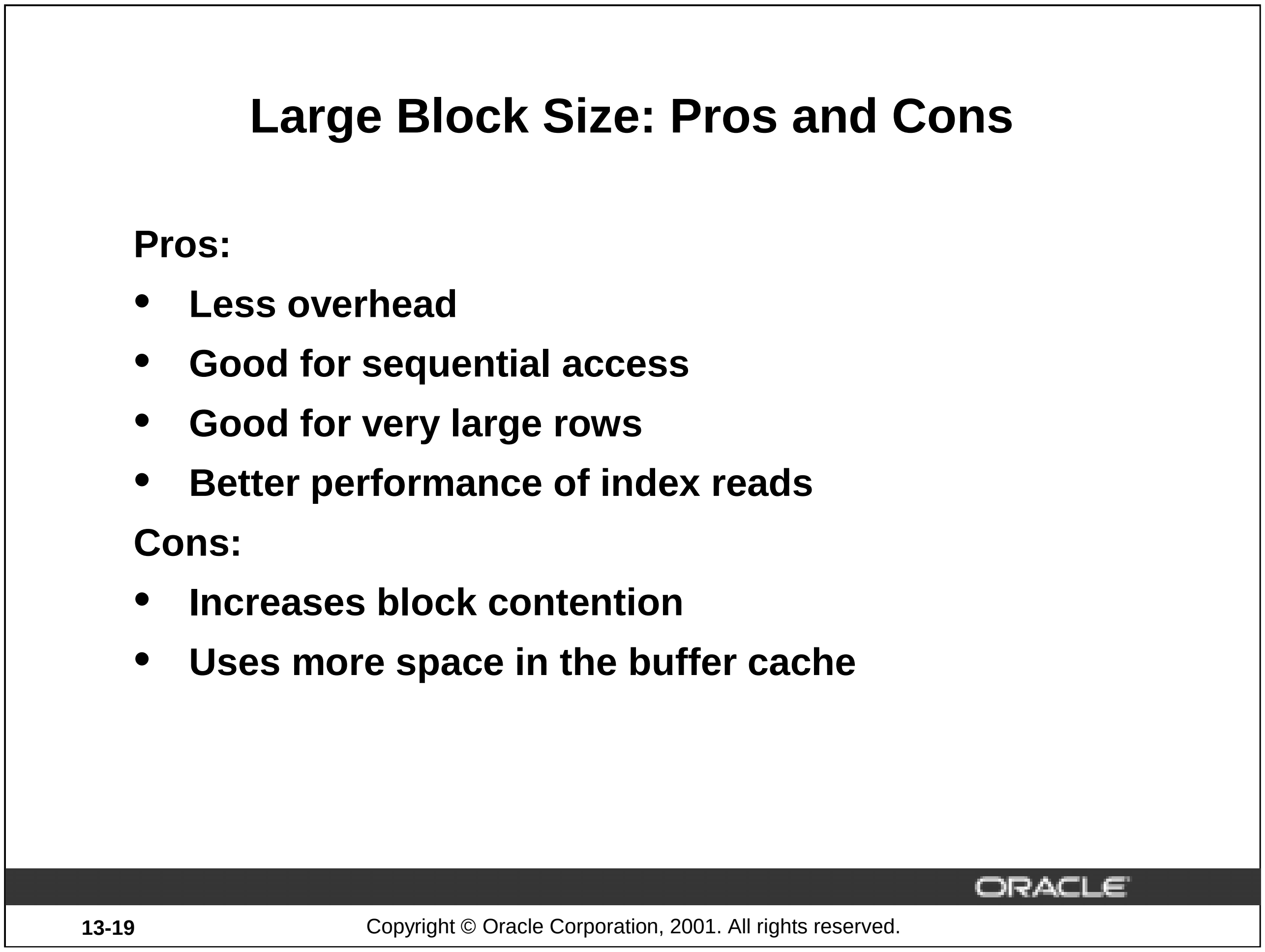
- Small blocks reduce block contention, because there are fewer rows per block.
- Small blocks are good for small rows.
- Small blocks are good for random access. Because it is unlikely that a block will be reused after it is read into memory, a smaller block size makes more efficient use of the buffer cache. This is especially important when memory resources are scarce, because the size of the database buffer cache is limited.

Disadvantages

- Small blocks have relatively large overhead.
- You may end up storing only a small number of rows per block, depending on the size of the row. This may cause additional I/Os.
- This may cause more index blocks to be read.

Performance

The block size chosen does affect performance. For random access to a large object, as in an OLTP environment, small blocks are favored.



Large Block Size: Pros and Cons

Pros:

- **Less overhead**
- **Good for sequential access**
- **Good for very large rows**
- **Better performance of index reads**

Cons:

- **Increases block contention**
- **Uses more space in the buffer cache**

ORACLE

13-19

Large Oracle Blocks

Advantages

- There is less overhead and thus more room to store useful data.
- Large blocks are good for sequential reads.
- Large blocks are good for very large rows.
- Larger blocks improve the performance of index reads. The larger blocks can hold more index entries in each block, which reduces the number of levels in large indexes. Fewer index levels mean fewer I/Os when traversing the index branches.

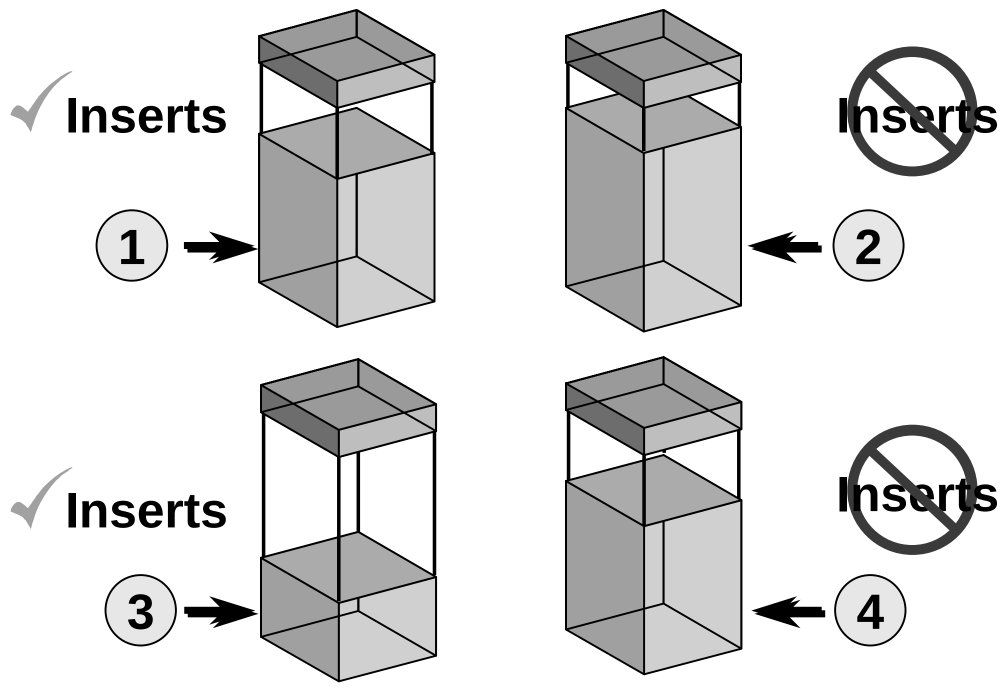
Disadvantages

- A large block size is not good for index blocks used in an OLTP environment, because they increase block contention on the index leaf blocks.
- Space in the buffer cache is wasted if you randomly access small rows and have a large block size. For example, with an 8 KB block size and a 50-byte row size, you waste 7,950 bytes in the buffer cache when doing a random access.

Performance

If you resize database blocks and there is no additional memory, you also need to reset `DB_BLOCK_BUFFERS`. This affects the cache hit percentage. The block size chosen does affect performance. Sequential access to large amounts of data, as in a DSS environment, prefers large blocks.

PCTFREE and PCTUSED



ORACLE

How PCTFREE and PCTUSED Work Together

Two space management parameters, PCTFREE and PCTUSED, enable you to control the use of free space within all the data blocks of a segment. You specify these parameters when creating or altering a table or cluster (which has its own data segment). You can also specify the `PCTFREE` storage parameter when creating or altering an index (which has its own index segment).

The `PCTFREE` parameter sets the minimum percentage of a data block to be reserved as free space for possible updates to rows that already exist in that block.

The `PCTUSED` parameter sets the minimum percentage of a block that can be used for row data plus overhead before new rows are added to the block.

As an example, if you issue a `CREATE TABLE` statement with set to `PCTFREE` 20, the Oracle server reserves 20% of each data block in the data segment of this table for updates to the existing rows in each block. The used space in the block can grow (1) until the row data and overhead total 80% of the total block size. Then the block is removed from the free list to prevent additional inserts (2).

The Effects of DML on PCTFREE

After you issue a DELETE or UPDATE statement, the Oracle server processes the statement and checks to see whether the space being used in the block is now less than PCTUSED. If it is, the block goes to the beginning of the free list. When the transaction is committed, free space in the block becomes available for other transactions (3).

After a data block is filled to the PCTFREE limit again (4), the Oracle server again considers the block unavailable for the insertion of new rows until the percentage of that block falls below the PCTUSED parameter.

DML Statements, PCTFREE, and PCTUSED

Two types of statements can increase the free space of one or more data blocks: DELETE statements and UPDATE statements that update existing values to values that use less space.

Released space in a block may not be contiguous; for example, when a row in the middle of the block is deleted. The Oracle server coalesces the free space of a data block only when:

- An INSERT or UPDATE statement attempts to use a block that contains enough free space to contain a new row piece.
- The free space is fragmented so that the row piece cannot be inserted in a contiguous section of the block.

The Oracle server performs this compression only in such situations, because otherwise the performance of a database system would decrease due to the continuous compression of the free space in data blocks.

Guidelines for PCTFREE and PCTUSED

PCTFREE

- **Default is 10**
- **Zero if no UPDATE activity**
- **PCTFREE = 100 × UPD / (Average row length)**

PCTUSED

- **Default is 40**
- **Set if rows are deleted**
- **PCTUSED = 100 – PCTFREE – 100 × Rows × (Average row length) / Block size**

Guidelines for Setting the Values of PCTFREE and PCTUSED

PCTFREE = 100 × UPD / (Average row length)

PCTUSED = 100 – PCTFREE – 100 × Rows × (Average row length) / block size

where:

UPD = Average amount added by updates, in bytes. This is determined by subtracting the average row length of the insertcurrent average row length.

Average row length = Get this value from the AVG_ROW_LEN column of DBA_TABLES, which you retrieve after running ANALYZE

Rows = Number of rows to be deleted before free list maintenance occurs

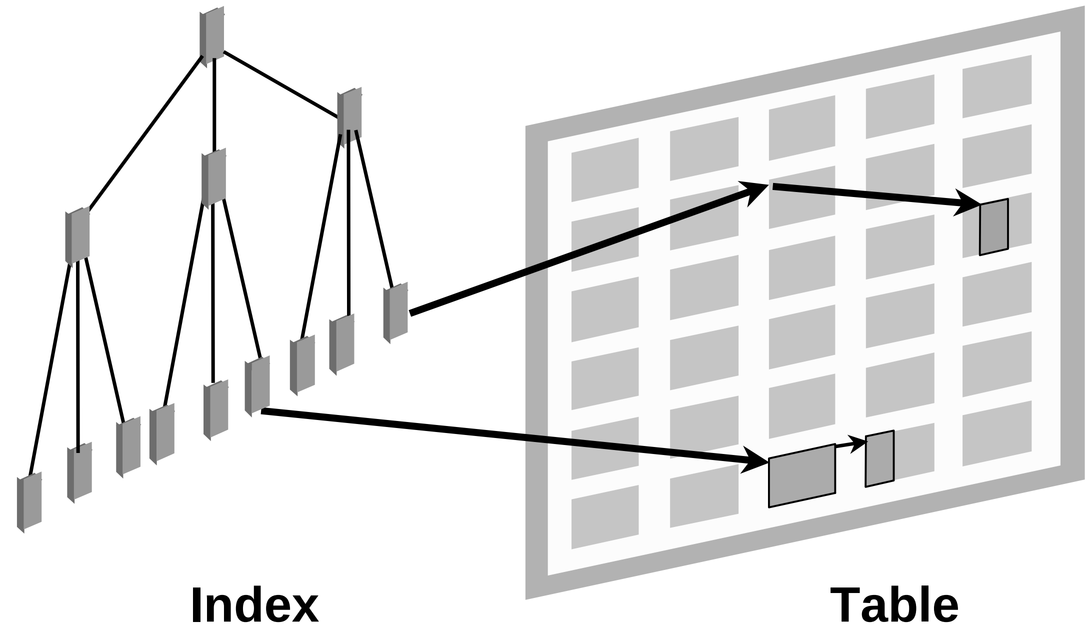
The formula for PCTUSED allows inserts into the table when there is enough space in the block for row updates and for at least one more row.

PCTUSED is relevant only in tables that undergo deletes. In many tables you may be able to pack rows into blocks more tightly by setting PCTUSED to be higher than the default (40%).

For tables with many inserts, changing PCTUSED may improve block storage performance. Blocks that are densely populated can cause free list contention, but fewer blocks are required, tables are smaller, and read operations are faster. Choose a value for PCTUSED in order to find a balance between those two different kinds of performance concerns.

If you change PCTFREE and PCTUSED for existing tables, there is no immediate impact on blocks. However, future DML activity uses the new rules for tables.

Migration and Chaining



Migration and Chaining

In two circumstances, the data for a row in a table may be too large to fit into a single data block.

- In the first case, called *chaining,* the row is too large to fit into an empty data block. In this case, the Oracle server stores the data for the row in a chain of one or more data blocks. Chaining can occur when the row is inserted or updated. Row chaining usually occurs with large rows, such as rows that contain a large object (LOB). Row chaining in these cases is unavoidable, unless a larger block size is used.
- However, in the second case, called *migration,* an `UPDATE` statement increases the amount of data in a row so that the row no longer fits in its data block. The Oracle server tries to find another block with enough free space to hold the entire row. If such a block is available, the Oracle server moves the entire row to the new block. The Oracle server keeps the original row piece of a migrated row to point to the new block containing the actual row; the row ID of a migrated row does not change. Indexes are not updated, so they point to the original row location.

Migration and chaining have a negative effect on performance:

- `INSERT` and `UPDATE` statements that cause migration and chaining perform poorly, because they perform additional processing.
- Queries that use an index to select migrated or chained rows must perform additional I/Os.

Migration is caused by `PCTFREE` being set too low; there is not enough room in the block for updates. To avoid migration, all tables that are updated should have their `PCTFREE` set so that there is enough space within the block for updates.

Detecting Migration and Chaining

Detect migration and chaining using the ANALYZE command:

```
SQL> ANALYZE TABLE sales.order_hist COMPUTE STATISTICS;
Table analyzed.
```

```
SQL> SELECT num_rows, chain_cnt FROM dba_tables
 2   WHERE table_name='ORDER_HIST';
 NUM_ROWS CHAIN_CNT
--------- ---------
      168       102
```

Detect migration and chaining from STATSPACK:

```
Statistic                  Total Per transaction ...
------------------------- ----- -------------- ...
table fetch continued row   495            .02 …
```

The ANALYZE ... COMPUTE STATISTICS Command

You can identify the existence of migrated and chained rows in a table or cluster by using the ANALYZE command with the COMPUTE STATISTICS option. This command counts the number of migrated and chained rows and places this information into the chain_cnt column of DBA_TABLES.

The Num_rows column provides the number of rows stored in the analyzed table or cluster. Compute the ratio of chained and migrated rows to the number of rows to decide whether migrated rows need to be eliminated.

The Table Fetch Continued Row Statistic

You can also detect migrated or chained rows by checking the Table Fetch Continued Row statistic in V$SYSSTAT or in the STATSPACK report under “Instance Activity Stats for DB.”

Guidelines

Increase PCTFREE to avoid migrated rows. If you leave more free space available in the block for updates, the row has room to grow. You can also reorganize (re-create) tables and indexes with a high deletion rate.

Selecting Migrated Rows

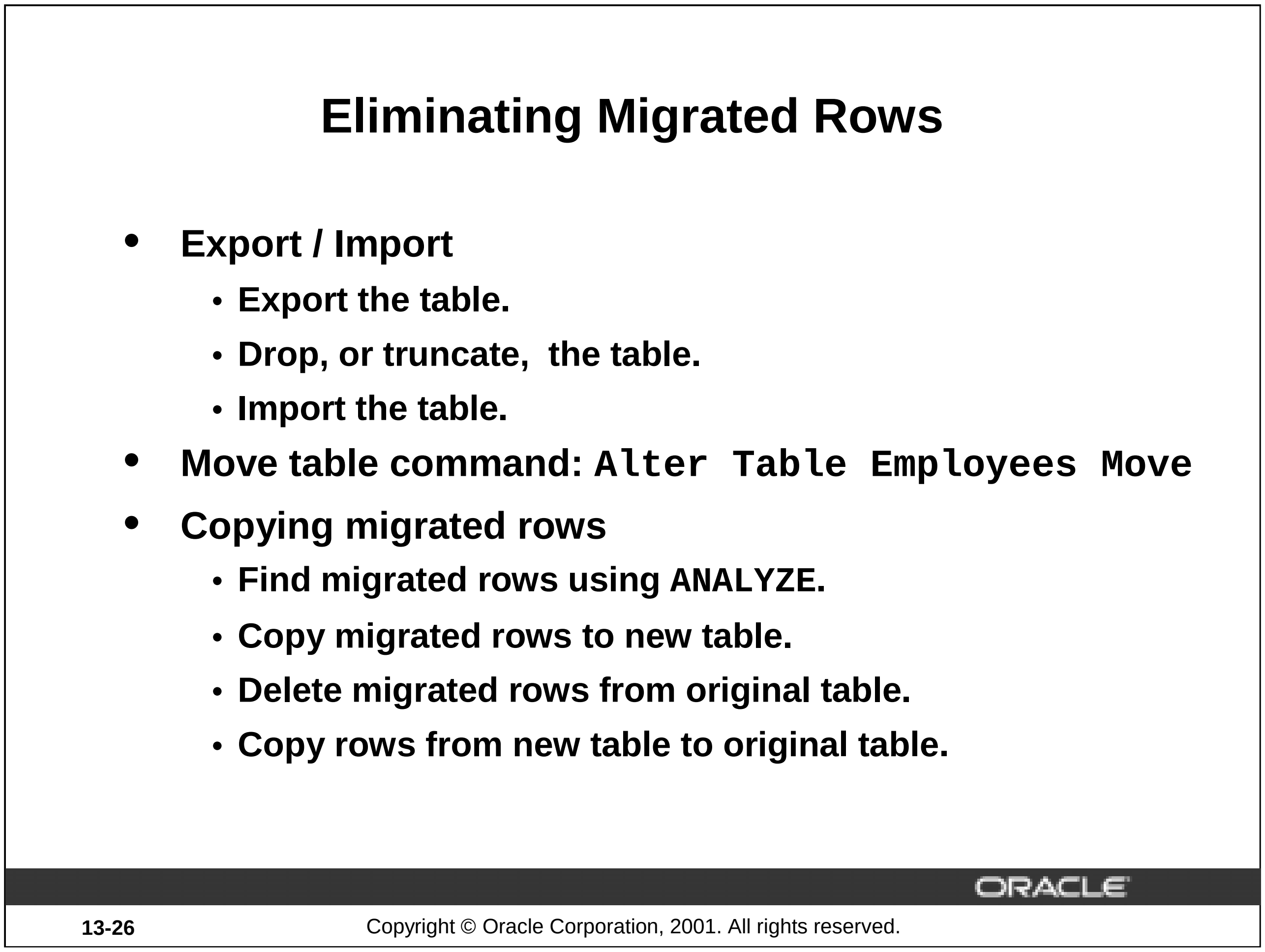
```
SQL> ANALYZE TABLE sales.order_hist LIST CHAINED ROWS;
Table analyzed.

SQL> SELECT  owner_name, table_name, head_rowid
  2    FROM  chained_rows
  3    WHERE table_name = 'ORDER_HIST';
OWNER_NAME  TABLE_NAME  HEAD_ROWID
----------  ----------  ------------------
SALES       ORDER_HIST  AAAAluAAHAAAAA1AAA
SALES       ORDER_HIST  AAAAluAAHAAAAA1AAB


...
```

ANALYZE ... LIST CHAINED ROWS

You can identify migrated and chained rows in a table or cluster by using the `ANALYZE` command with the `LIST CHAINED ROWS` option. This command collects information about each migrated or chained row and places this information into a specified output table. To create the table that holds the chained rows, execute the `utlchain.sql` script:

```
create table CHAINED_ROWS (
  owner_name                    varchar2(30),
  table_name                    varchar2(30),
  cluster_name                  varchar2(30),
  partition_name                varchar2(30),
  head_rowid                    rowid,
  analyze_timestamp             date );
```

If you create this table manually, it must have the same column names, data types, and sizes as the `chained_rows` table.



Eliminating Migrated Rows

- **Export / Import**
  - **Export the table.**
  - **Drop, or truncate, the table.**
  - **Import the table.**
- **Move table command: `Alter Table Employees Move`**
- **Copying migrated rows**
  - **Find migrated rows using `ANALYZE`.**
  - **Copy migrated rows to new table.**
  - **Delete migrated rows from original table.**
  - **Copy rows from new table to original table.**

Eliminating Migrated Rows

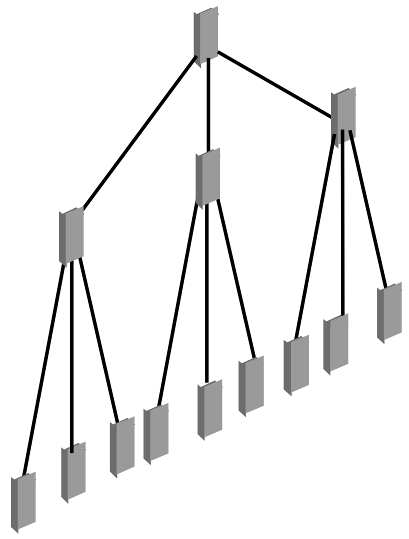
You can eliminate migrated rows using this SQL*Plus script:

```
/* Get the name of the table with migrated rows */
accept table_name prompt ’Enter the name of the table with
  migrated rows: ’
/* Clean up from last execution */
set echo off
drop table migrated_rows;
drop table chained_rows;

/* Create the CHAINED_ROWS table */
@?/rdbms/admin/utlchain
set echo on
spool fix_mig
/* List the chained & migrated rows */
analyze table &table_name list chained rows;
```

```
/* Copy the chained/migrated rows to another table */
create table migrated_rows as
  select  orig.* from  &table_name orig, chained_rows cr
    where orig.rowid = cr.head_rowid
    and   cr.table_name = upper('&table_name');

/* Delete the chained/migrated rows from the original table */
delete from &table_name
where rowid in ( select  head_rowid from  chained_rows );

/* Copy the chained/migrated rows back into the original table
*/
insert into &table_name select  * from  migrated_rows;

spool off
```

When using this script, you must disable any foreign key constraints that would be violated when the rows are deleted.

Index Reorganization



- Indexes on volatile tables are a performance problem.
- Only entirely empty index blocks go to the free list.
- If a block contains only one entry, it must be maintained.
- You may need to rebuild indexes.

13-28

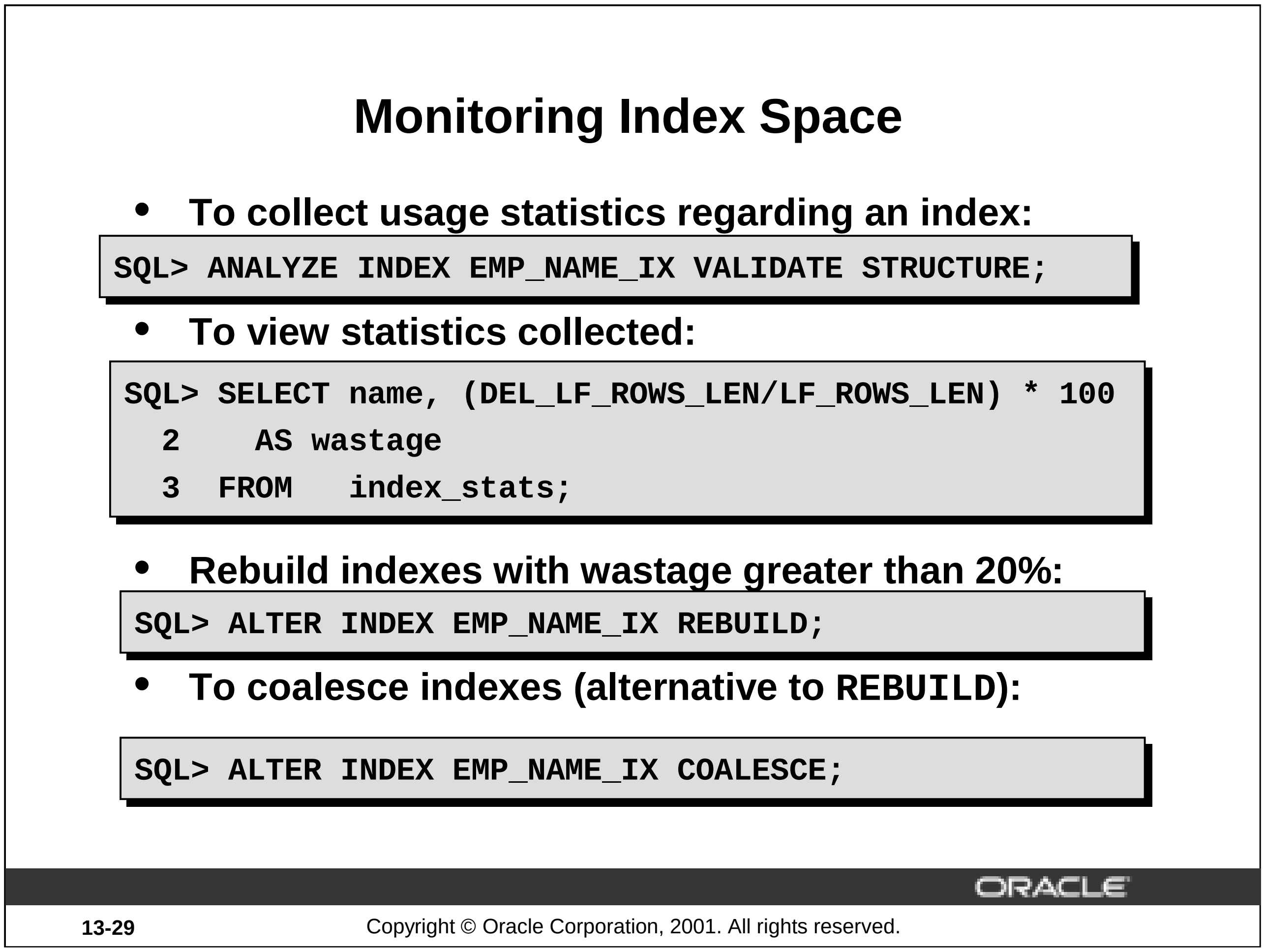
Index Reorganization

Indexes on volatile tables can also present a performance problem.

In data blocks, the Oracle server replaces deleted rows with inserted ones; however, in index blocks, the Oracle server orders entries sequentially. Values must go in the correct block, together with others in the same range.

Many applications insert in ascending index order and delete older values. But even if a block contains only one entry, it must be maintained. In this case, you may need to rebuild your indexes regularly.

If you delete all the entries from an index block, the Oracle server puts the block back on the free list.



Monitoring Index Space

- To collect usage statistics regarding an index:

```
SQL> ANALYZE INDEX EMP_NAME_IX VALIDATE STRUCTURE;
```

- To view statistics collected:

```
SQL> SELECT name, (DEL_LF_ROWS_LEN/LF_ROWS_LEN) * 100
  2    AS wastage
  3  FROM   index_stats;
```

- Rebuild indexes with wastage greater than 20%:

```
SQL> ALTER INDEX EMP_NAME_IX REBUILD;
```

- To coalesce indexes (alternative to `REBUILD`):

```
SQL> ALTER INDEX EMP_NAME_IX COALESCE;
```

13-29

Monitoring Index Space

You can monitor the space used by indexes by using the command:

SQL> ANALYZE INDEX index_name VALIDATE STRUCTURE;

By querying the INDEX_STATS view, values for the following columns can be found :

• `Lf_rows`	- Number of values currently in the index
• `Lf_rows_len`	- Sum of all the length of values, in bytes
• `Del_lf_rows`	- Number of values deleted from the index
• `Del_lf_rows_len`	- Length of all deleted values

Note: The INDEX_STATS view contains the last analyzed index. The view is session-specific. If you connect to another session under the same user, the view is populated with the index analyzed by the session querying the view.

Deciding Whether to Rebuild or Coalesce an Index

REBUILD	Coalesce
Quickly moves index to another tablespace	Cannot move index to another tablespace
Higher costs: requires more disk space	Lower costs: does not require more disk space
Creates new tree, shrinks height if applicable	Coalesces leaf blocks within same branch of tree
Enables you to quickly change storage and tablespace parameters without having to drop the original index.	Quickly frees up index leaf blocks for use.

Rebuilding Indexes

You may decide to rebuild an index if the deleted entries represent 20% or more of the current entries, although this depends on your application and priorities. You can use the query on the previous slide to find the ratio. Use the `ALTER INDEX REBUILD` statement to reorganize or compact an existing index or to change its storage characteristics. The `REBUILD` statement uses the existing index as the basis for the new index. All index storage commands are supported, such as `STORAGE` (for extent allocation), `TABLESPACE` (to move the index to a new tablespace), and `INITRANS` (to change the initial number of entries). When building an index, you can also use the following keywords to reduce the time it takes to rebuild:

- `PARALLEL` or `NOPARALLEL (NOPARALLEL` is the default)
- `RECOVERABLE` or `UNRECOVERABLE` (`RECOVERABLE` is the default):
  - `UNRECOVERABLE` is faster because it does not write redo log entries during the index creation or rebuild.
  - This clause does not set a permanent attribute of the object, but is only effective during the creation operation; thus, the object does not appear in the dictionary.
  - This attribute cannot be updated.
  - This attribute is usable only at object creation or rebuild.
  - This attribute implies `LOGGING` by default, which means that further inserts are logged.
  - The index is not recoverable if it needs to be re-created during a recover operation.
- `LOGGING` or `NOLOGGING`
  - `NOLOGGING` is faster because it does not write any redo log entries during the index life until `NOLOGGING` is changed to `LOGGING` .
  - It is a permanent attribute and thus appears in the dictionary.
  - It can be updated with the `ALTER INDEX NOLOGGING/LOGGING` command at any time.

Note: `UNRECOVERABLE` and `LOGGING` are not compatible.

The `UNRECOVERABLE` option can be used in early versions of Oracle8. It is kept for backwards compatability only and will be phased out eventually.

To duplicate the semantics of `UNRECOVERABLE` clause, create an object with the `NOLOGGING` option and then alter it, specifying `LOGGING` . To duplicate the semantics of a `RECOVERABLE` clause, create an object with the `LOGGING` option.

`ALTER INDEX REBUILD` is usually faster than dropping and re-creating an index because it uses the fast full scan feature. It thus reads all the index blocks, using multiblock I/O, then discards the branch blocks. Oracle8*i* introduced a method of creating an index or re-creating an existing index while allowing concurrent operations on the base table. This helps achieve demanding availability goals by allowing maintenance operations to be performed online with no down time.

Monitoring Index Usage

- **Gathering statistics using an Oracle supplied package:**

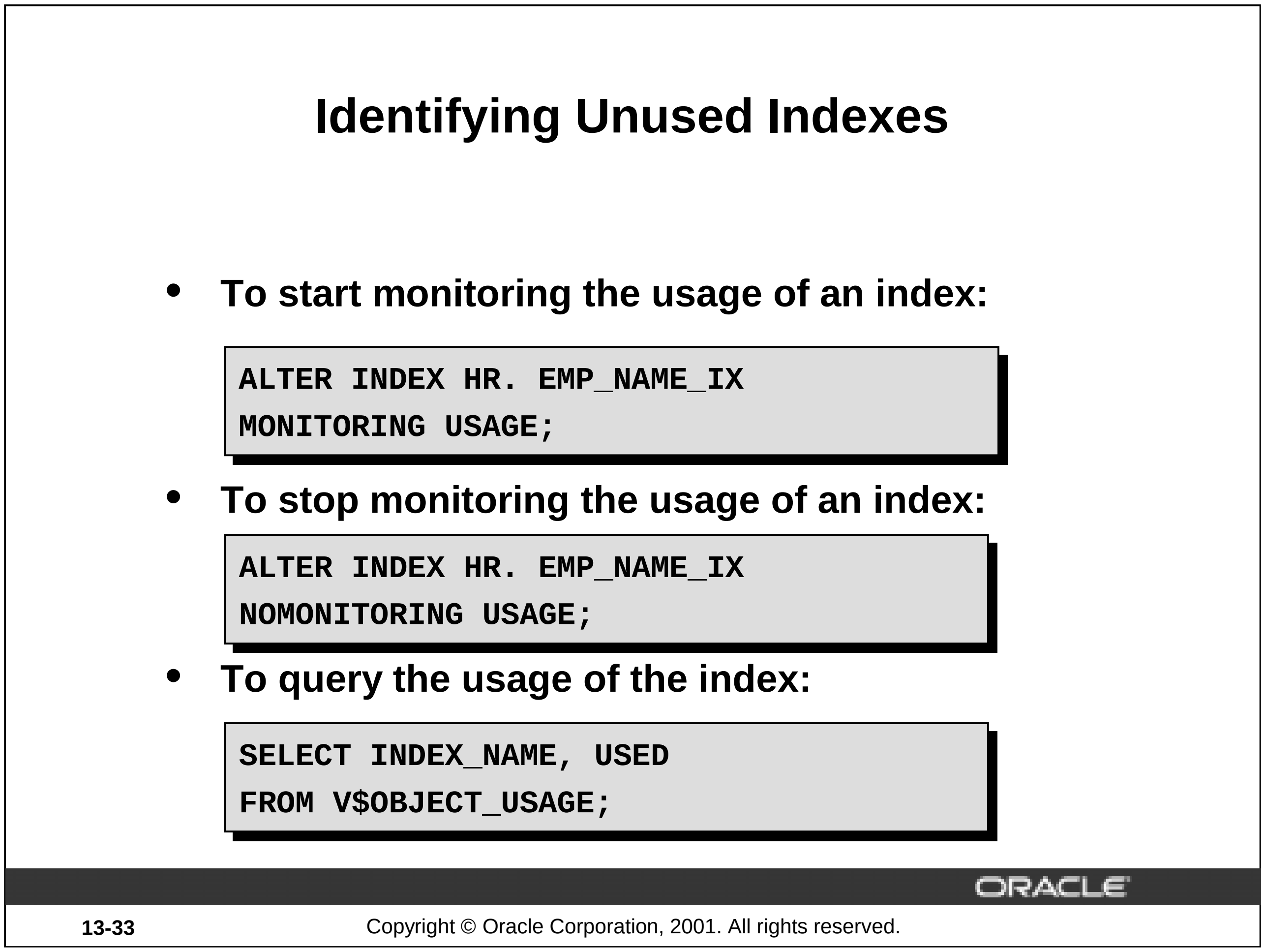
```
SQL> Execute  DBMS_STATS.GATHER_INDEX_STATS
                         (‘HR’,’LOC_COUNTRY_IX’);
```

- **Gathering statistics at index creation:**

```
SQL> Create index hr.loc_country_ix…
       ……………………
       compute statistics;
```

- **Gathering statistics when rebuilding an index:**

```
SQL> Alter index hr.loc_country_ix rebuild compute
statistics;
```



Identifying Unused Indexes

- **To start monitoring the usage of an index:**

```
ALTER INDEX HR. EMP_NAME_IX
MONITORING USAGE;
```

- **To stop monitoring the usage of an index:**

```
ALTER INDEX HR. EMP_NAME_IX
NOMONITORING USAGE;
```

- **To query the usage of the index:**

```
SELECT INDEX_NAME, USED
FROM V$OBJECT_USAGE;
```

13-33

Identifying Unused Indexes

Beginning with Oracle9*i*, statistics about the usage of an index can be gathered and displayed in `V$OBJECT_USAGE`. If the information gathered indicates that an index is never used, the index can be dropped. In addition, eliminating unused indexes cuts down on the overhead for DML operations, thus improving performance. Each time the `MONITORING USAGE` clause is specified, `V$OBJECT_STATS` is reset for the specified index. The previous information is cleared, and a new start time is recorded.

`V$OBJECT_USAGE` Columns

• index_name	- The index name
• table_name	- The corresponding table
• monitoring	- Indicates whether monitoring is “ON or OFF”
• used	- Indicates (YES or NO) the index has been used during the monitoring time
• start_monitoring	- Time at which monitoring began on index
• stop_monitoring	- Time at which monitoring stopped on index

Summary

In this lesson, you should have learned to do the following:

- Describe the correct usage of extents and Oracle blocks
- Explain space usage and the high-water mark
- Determine the high-water mark
- Describe the use of Oracle Block parameters
- Recover space from sparsely populated segments
- Describe and detect chaining and migration of Oracle blocks
- Perform index reorganization
- Monitor indexes to determine usage

Practice 13

1. Connect using sys/oracle as sysdba and query the tablespace_name and extent_management columns of DBA_TABLESPACES to determine which tablespaces are locally managed, and which are dictionary managed. Record which tablespaces are dictionary managed.
2. Alter user HR to have the TOOLS tablespace as the default.
3. Examine the v$system_event view and note the total_waits for the statistic enqueue.

   Note: On a production system you would be more likely to pickup the contention through the STATSPACK report.
4. Also examine the view v$enqueue_stat for eq_type 'ST' in order to determine the total_wait# for the ST enqueue, which is the space management enqueue.
5. Exit out of the SQLPlus session and change directory to $HOME/STUDENT/LABS. Run the script lab13_04.sh from the operating system prompt. This script will log five users onto the database simultaneously and then each user creates, then drops, tables. The tables each have many extents. The script must be run from the $HOME/STUDENT/LABS directory, or it will fail.
6. Connect as system/manager and again examine the view v$enqueue_stat for eq_type 'ST' in order to determine if the value of total_wait# for the ST enqueue, which is the space management enqueue.

   Note: Record the difference in the number of waits for the ST enqueue for extent management using a dictionary managed tablespace. This value is found by subtracting the first wait value (from practice 13-04) from the second wait value (from practice 13-06).
7. Create a new locally managed tablespace TEST, name the datafile test01.dbf, and place it in the directory $HOME/ORADATA/u06. Set the size to 120M, and a uniform extent size of 20k.
8. Alter the default tablespace of user hr to test.

   Note: The same steps are covered again. This time you are looking for the number of waits for the ST enqueue caused by locally managed tablespaces.
9. Examine, and record the initial total_wait# for 'ST' in the v$enqueue_stat view.
10. Exit out of the SQLPlus session and change directory to $HOME/STUDENT/LABS. Run the script lab13_04.sh from the operating system prompt. This script will log five users onto the database simultaneously and then each user creates, then drops, tables. The tables each have many extents. The script must be run from the $HOME/STUDENT/LABS directory, or it will fail.
11. Again examine, and record the final total_wait# for 'ST' in the v$enqueue_stat view.

    Note: Record the difference in the total_wait# for the ST enqueue for extent management using a locally managed tablespace. This value is found by subtracting the first wait value (from practice 13-09) from the second wait value (from practice 13-11). Compare the two results for the different tablespaces. The locally managed tablespace would be far less contentious with extent management since it is managing the space within the tablespace itself.

Practice 13 (continued)

12. Connect as user hr/hr and run the script $HOME/STUDENT/LABS/lab13_12.sql. This will create a similar table (NEW_EMP) as the employees table but with PCTFREE = 0. The table is then populated with data from the employees table.
13. Run analyze on the new_emp table and query the DBA_TABLES view to determine the value of chain_cnt for the new_emp table. Record this value.
14. Create an index new_emp_name_idx on the last_name column of the new_emp table. Place the index in the tablespace INDX. Then confirm the index's status in user_indexes.
15. Run the script $HOME/STUDENT/LABS/lab13_15.sql which will update the rows of the new_emp table. Analyze the new_emp table again and query the USER_TABLES view to get the new value of chain_cnt Record this value. Also check the status of the index new_emp_name_idx.
16. Resolve the migration caused by the previous update, by using the alter table move command. This will cause the index to become unusable, and should be rebuilt using the alter index rebuild command before re-analyzing the table new_emp. Confirm that the migration has been resolved by querying chain_cnt column in user_tables, and confirm that the index is valid by querying user_indexes.

Quick Reference

Context	**Reference**
Initialization parameters	None

Dynamic performance views	`V$SYSSTAT`
	`V$SESSTAT`
	`V$MYSTAT`

Data dictionary views	`USER_, ALL_, DBA_CLUSTERS`
	`USER_, ALL_, DBA_INDEXES`
	`USER_, ALL_, DBA_TABLES`
	`INDEX_STATS`

Commands	`CREATE TABLESPACE ... EXTENT MANAGEMENT LOCAL`
	`ALTER/CREATE INDEX/TABLE/CLUSTER`
	`ALTER INDEX ... REBUILD`
	`TRUNCATE`
	`ANALYZE ... COMPUTE STATISTICS`
	`ANALYZE ... LIST CHAINED ROWS`

Packaged procedures and functions	`DBMS_SPACE`
Scripts	`dbmsutil.sql`, `utlchain.sql`
Oracle Diagnostics Pack	Performance Manager

14

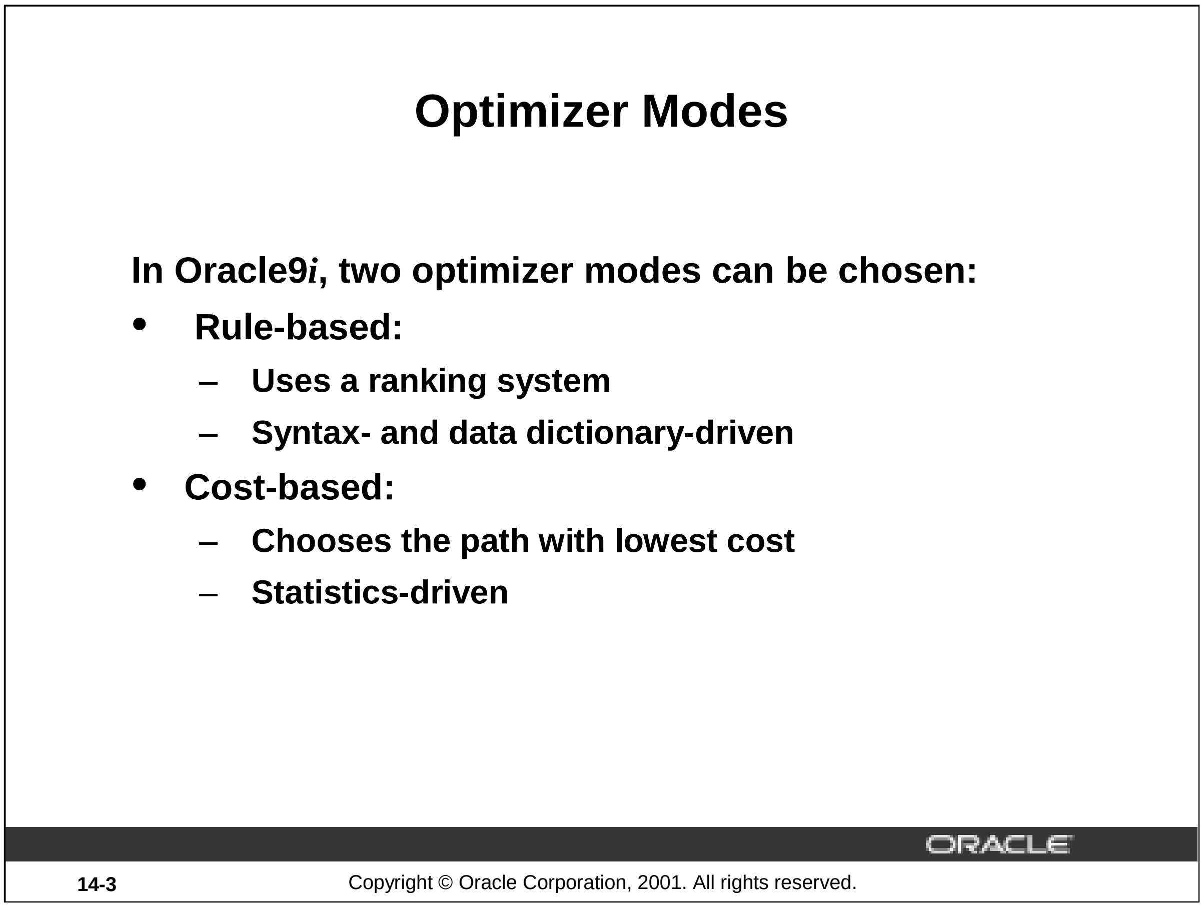
SQL Statement Tuning

Objectives

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

- Describe how the optimizer is used
- Explain the concept of plan stability
- Explain the use of stored outlines
- Describe how hints are used
- Use SQL Trace and `TKPROF`
- Collect statistics on indexes and tables
- Describe the use of histograms
- Copy statistics between databases

ORACLE



Optimizer Modes

In Oracle9*i*, two optimizer modes can be chosen:

- Rule-based:
  - Uses a ranking system
  - Syntax- and data dictionary-driven
- Cost-based:
  - Chooses the path with lowest cost
  - Statistics-driven

14-3

Optimizer Modes

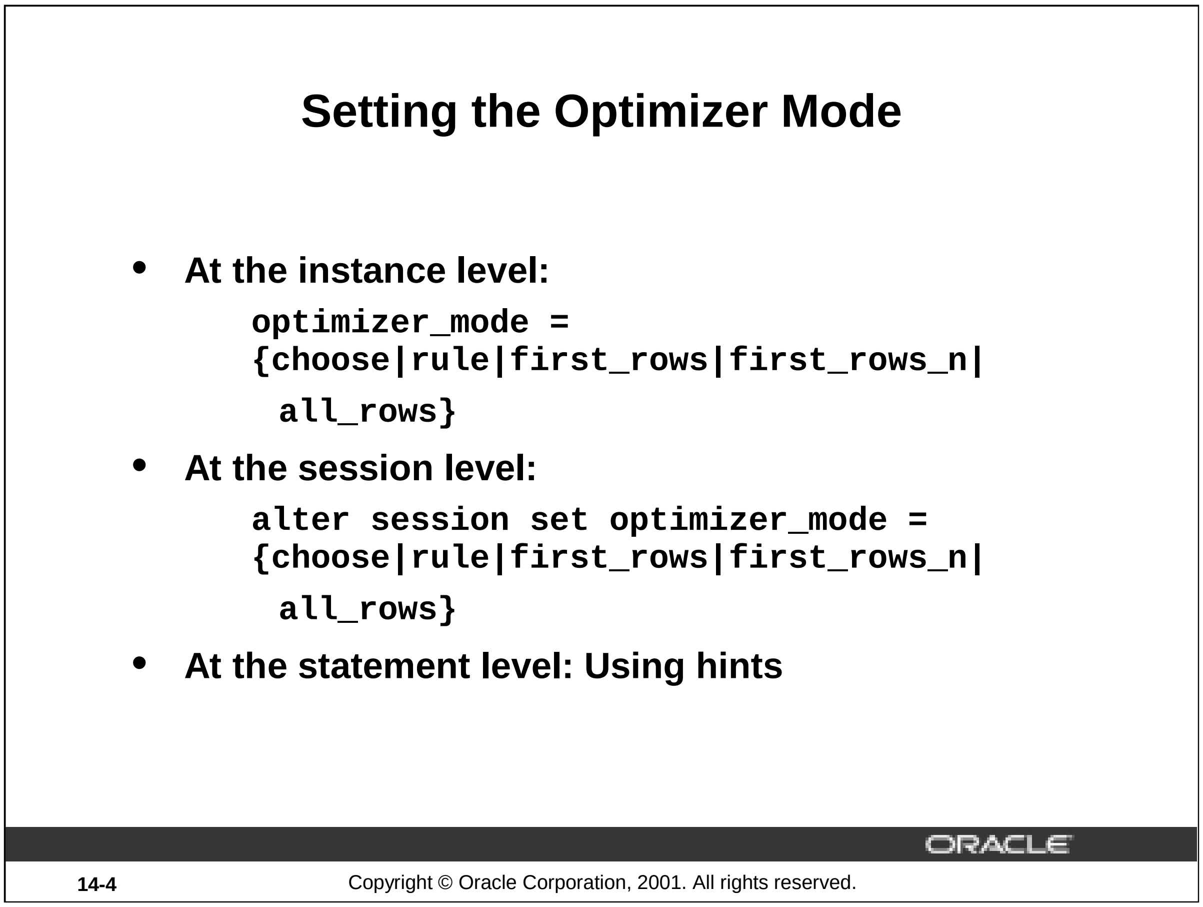
Rule-Based Optimization

In this mode, the server process chooses its access path to the data by examining the query. This optimizer has a complete set of rules for ranking access paths. Experienced Oracle developers often have a good understanding of these rules, and tune their SQL code accordingly. The rule-based optimizer is syntax-driven, in that it uses the statement syntax in combination with data dictionary information about the data structures to determine which execution plan to use. This optimizer mode is supported for backward compatibility with earlier releases of the Oracle server.

Cost-Based Optimization

In this mode, the optimizer examines each statement and identifies all possible access paths to the data. It then calculates the resource cost of each access path and chooses the least expensive one. The costing is based mainly on the number of logical reads. The cost-based optimizer is statistics-driven in that it uses statistics generated for the objects involved in the SQL statement to determine the most effective execution plan. The cost-based optimizer is used if any object in the SQL statement has had statistics generated for it. You should use this optimizer mode for new applications, particularly if they use the Parallel Query feature.

Note: For additional information about the ranking used by the rule-based optimizer, refer to the *Oracle9i Database Concepts* manual.



Setting the Optimizer Mode

- **At the instance level:**

```
optimizer_mode =
{choose|rule|first_rows|first_rows_n|
 all_rows}
```

- **At the session level:**

```
alter session set optimizer_mode =
{choose|rule|first_rows|first_rows_n|
 all_rows}
```

- **At the statement level: Using hints**

ORACLE

14-4 Copyright © Oracle Corporation, 2001. All rights reserved.

Setting the Optimizer Mode

The optimizer mode can be set at the:

- Instance level by using the `OPTIMIZER_MODE` parameter
- Session level, by using the ALTER SESSION command
- Statement level, by using hints

The DBA is responsible for setting the `OPTIMIZER_MODE` parameter at the instance level, because this requires restarting the instance. Typically, application developers can set the `OPTIMIZER_MODE` at the session level, as well as use hints in SQL statements.

The `OPTIMIZER_MODE` Parameter

The default value is CHOOSE. This means that the optimizer uses the cost-based mode (`ALL_ROWS`) if statistics are available for at least one of the tables involved. Otherwise, it uses rule-based optimization.

Note: If any table involved has a degree of parallellization greater than 1, the default behavior is cost-based optimization as well.

The other possible values are `RULE`, `FIRST_ROWS`, `FIRST_ROWS_n` and `ALL_ROWS`. The first one forces rule-based optimization regardless the existence of any statistics. The last two represent different ways of using cost-based optimization. `FIRST_ROWS` minimizes immediate response time (possibly at the expense of overall response time), and `FIRST_ROWS_n` minimizes immediate response time for the first n rows (possibly at the expense of overall response time). The value of n can be 1, 10, 100 or 1000. `ALL_ROWS` minimizes total response time (throughput).

The `OPTIMIZER_MODE` Option at the Session Level

Developers can set this option using the ALTER SESSION command.

```
SQL> ALTER SESSION SET OPTIMIZER_MODE = value
```

Note: For backward compatibility, the `OPTIMIZER_GOAL` option of the ALTER SESSION command is still supported as an alternative for the `OPTIMIZER_MODE` option.

Optimizer Hints

You can code hints into a statement, as shown below:

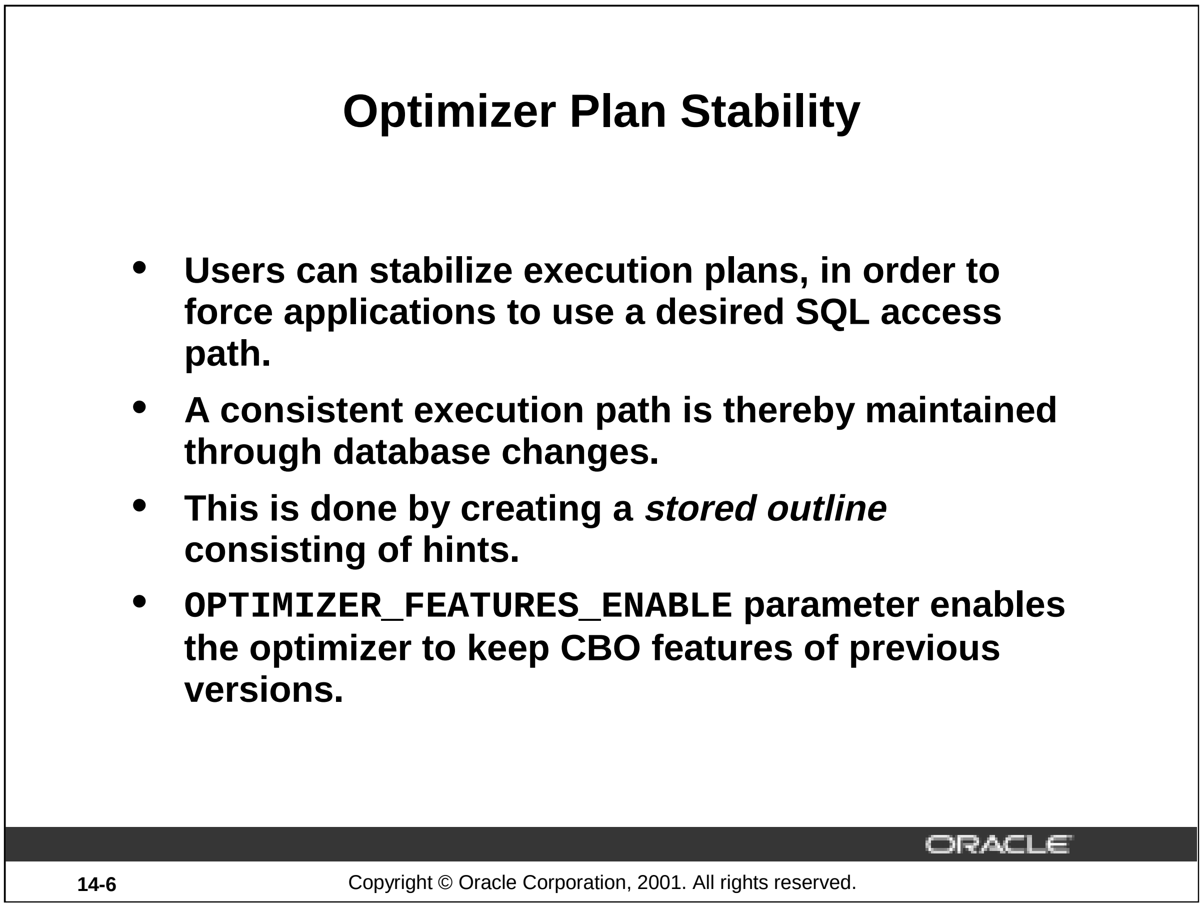
```
SQL>  SELECT /*+ FIRST_ROWS */
  2        *
  3  FROM  hr.employees;
```

Optimizer hints which influence the optimizer mode include `RULE`, `FIRST_ROWS`, `FIRST_ROWS_n` and `ALL_ROWS`.

Note: Refer to the *Oracle9i Performance Guide and Reference* manual for a listing of all available hints.

Precedence Rules

Hints always override session level-settings, and session-level settings always override instance-level settings.



Optimizer Plan Stability

- **Users can stabilize execution plans, in order to force applications to use a desired SQL access path.**
- **A consistent execution path is thereby maintained through database changes.**
- **This is done by creating a *stored outline* consisting of hints.**
- **`OPTIMIZER_FEATURES_ENABLE` parameter enables the optimizer to keep CBO features of previous versions.**

14-6

Optimizer Plan Stability

For every query, the optimizer prepares a tree of operations called an execution plan that defines the order and methods of operation that the server follows to execute the statement.

Because the optimizer may have incomplete information, sometimes the best possible plan is not chosen. In these cases, you may find it worthwhile to influence the optimizer's plan selection by rewriting the SQL statement, by using hints, or by using other tuning techniques. Once satisfied, you may want to ensure that the same tuned plan is generated whenever the same query is recompiled, even when factors that affect optimization may have changed.

Oracle9*i* provides the user with a means of stabilizing execution plans across Oracle releases, database changes, or other factors that normally cause an execution plan to change. You can create a stored outline containing a set of hints used by the optimizer to create an execution plan.

`OPTIMIZER_FEATURES_ENABLE`

This parameter allows a version of the Oracle server to run with the cost based optimizer features of an earlier version. This parameter should be left as its default, which is the current release. However, if the DBA wants to keep the previous CBO features while performing the upgrade then it can be set. Before doing so refer to the *Oracle9i Performance Guide and Reference* .

Plan Equivalence

- **SQL statement text must match the text in a stored outline.**
- **Plans are maintained through:**
  - **New Oracle versions**
  - **New statistics on objects**
  - **Initialization parameter changes**
  - **Database reorganization**
  - **Schema changes**

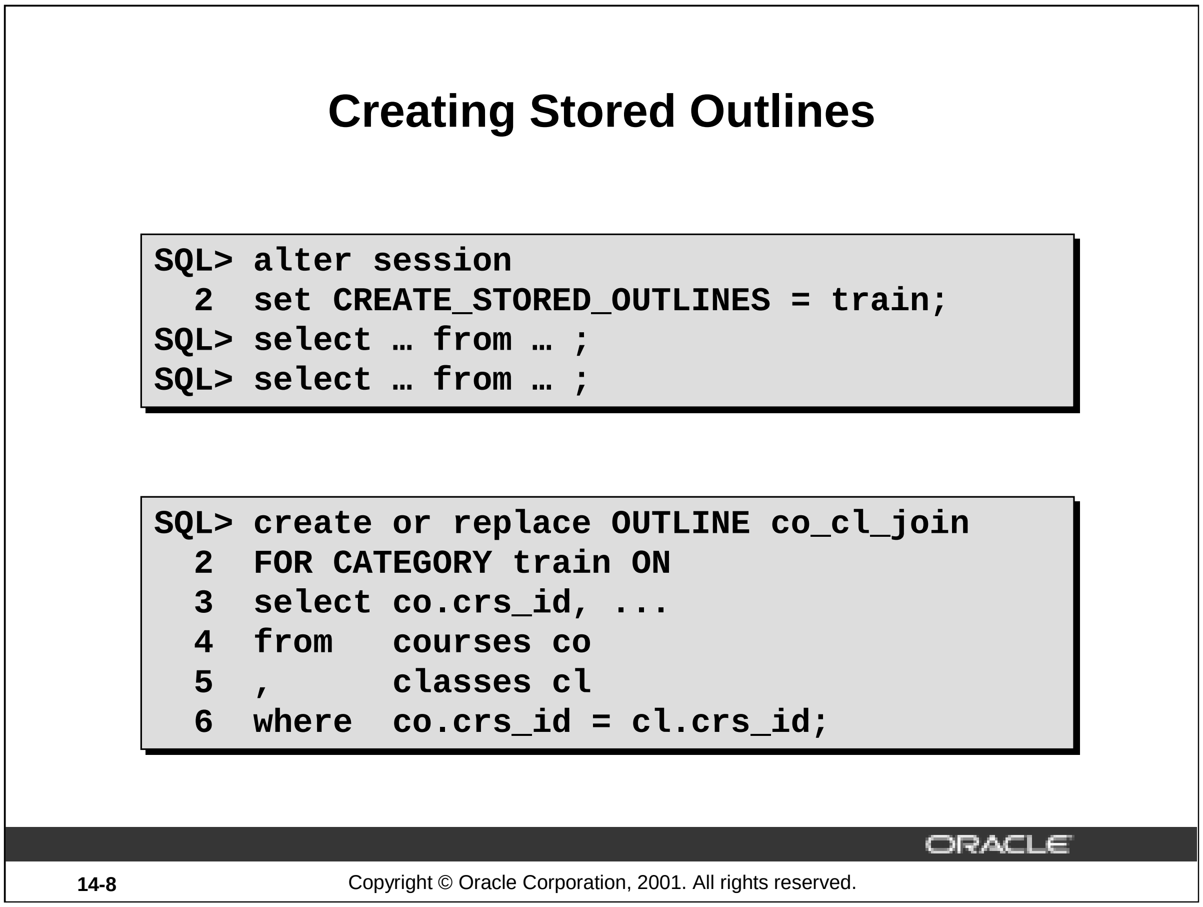
SQL Statement Equivalence

Plan stability relies on exact textual matching of queries when determining whether a query has a stored outline. This is the same matching criteria used to determine whether an execution plan in the shared pool can be reused.

Stored outlines rely partially on hints that the optimizer uses to achieve stable execution plans. Therefore, the degree to which plans remain equivalent is dependent on the capabilities of the hints the plans use. The execution steps included in a stored outline include row access methods, join order, join methods, distributed accesses, and view/subquery merging. Distributed access does not include the execution plan on the remote node.

Plan Stability

These plans are maintained through many types of database and instance changes. Therefore, if you develop applications for mass distribution, you can use stored outlines to ensure that all your customers access the same execution plans. For example, if a schema is changed by adding an index, the stored outline can prevent the use of the new index.



Creating Stored Outlines

```
SQL> alter session
  2  set CREATE_STORED_OUTLINES = train;
SQL> select … from … ;
SQL> select … from … ;
```

```
SQL> create or replace OUTLINE co_cl_join
  2  FOR CATEGORY train ON
  3  select co.crs_id, ...
  4  from   courses co
  5  ,      classes cl
  6  where  co.crs_id = cl.crs_id;
```

Creating Stored Outlines

The server can create outlines automatically or you can create them for specific SQL statements. Outlines use the cost-based optimizer, because they rely on hints.

Categories

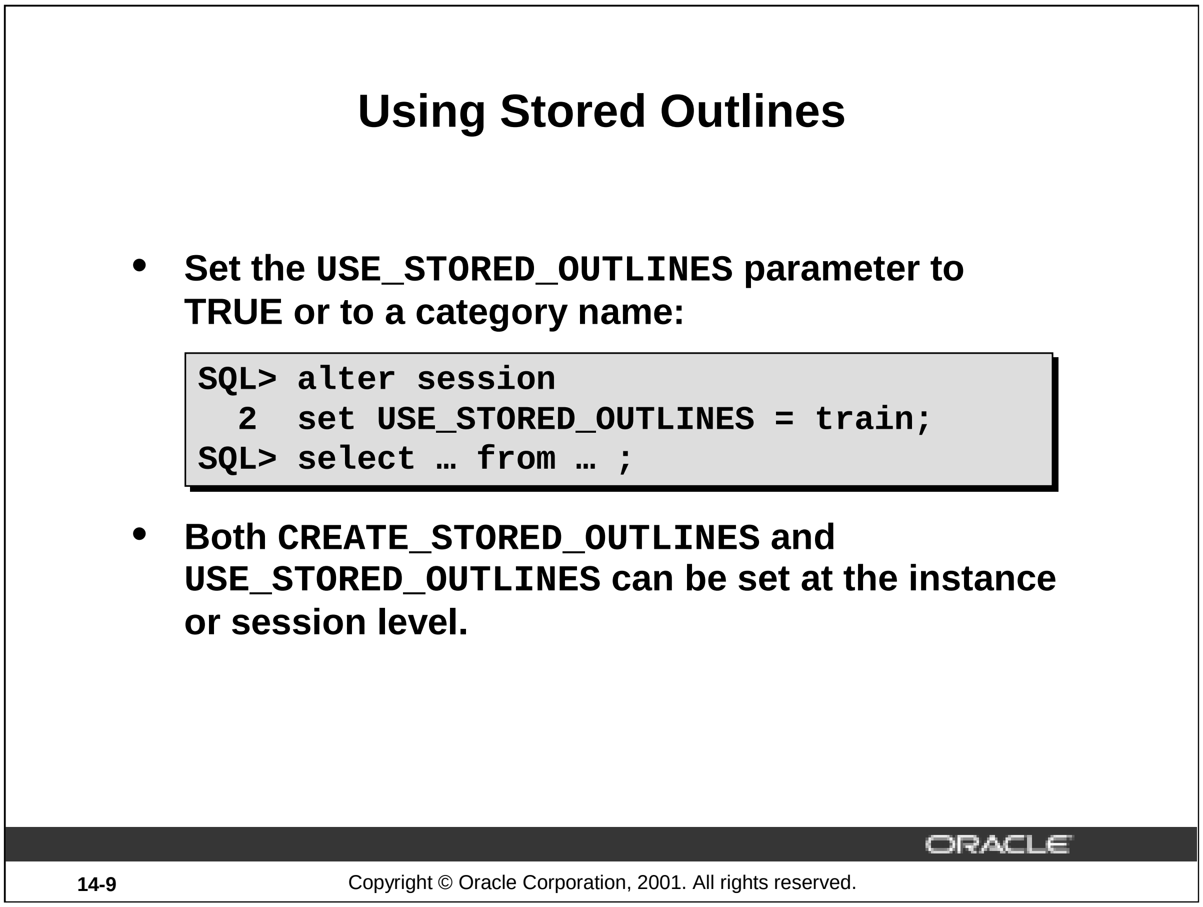
Stored outlines can be grouped by categories. The same SQL statement can have a stored outline in more than one category. For example, you may want to have an OLTP category and a DSS category. If a category name is omitted, outlines are placed in the DEFAULT category.

`CREATE_STORED_OUTLINES` Parameter

Oracle creates stored outlines automatically for all executed SQL statements when you set `CREATE_STORED_OUTLINES` to TRUE or to a category name. When set to TRUE, the DEFAULT category is used. You can deactivate the process by setting the parameter to FALSE. When this parameter is used, the outline names are also generated automatically.

CREATE OUTLINE Command

You can also create stored outlines for a specific statement by using the CREATE OUTLINE command. One advantage of this approach is that you can specify a name for the stored outline.



Using Stored Outlines

- **Set the `USE_STORED_OUTLINES` parameter to TRUE or to a category name:**

```
SQL> alter session
  2  set USE_STORED_OUTLINES = train;
SQL> select … from … ;
```

- **Both `CREATE_STORED_OUTLINES` and `USE_STORED_OUTLINES` can be set at the instance or session level.**

Using Stored Outlines

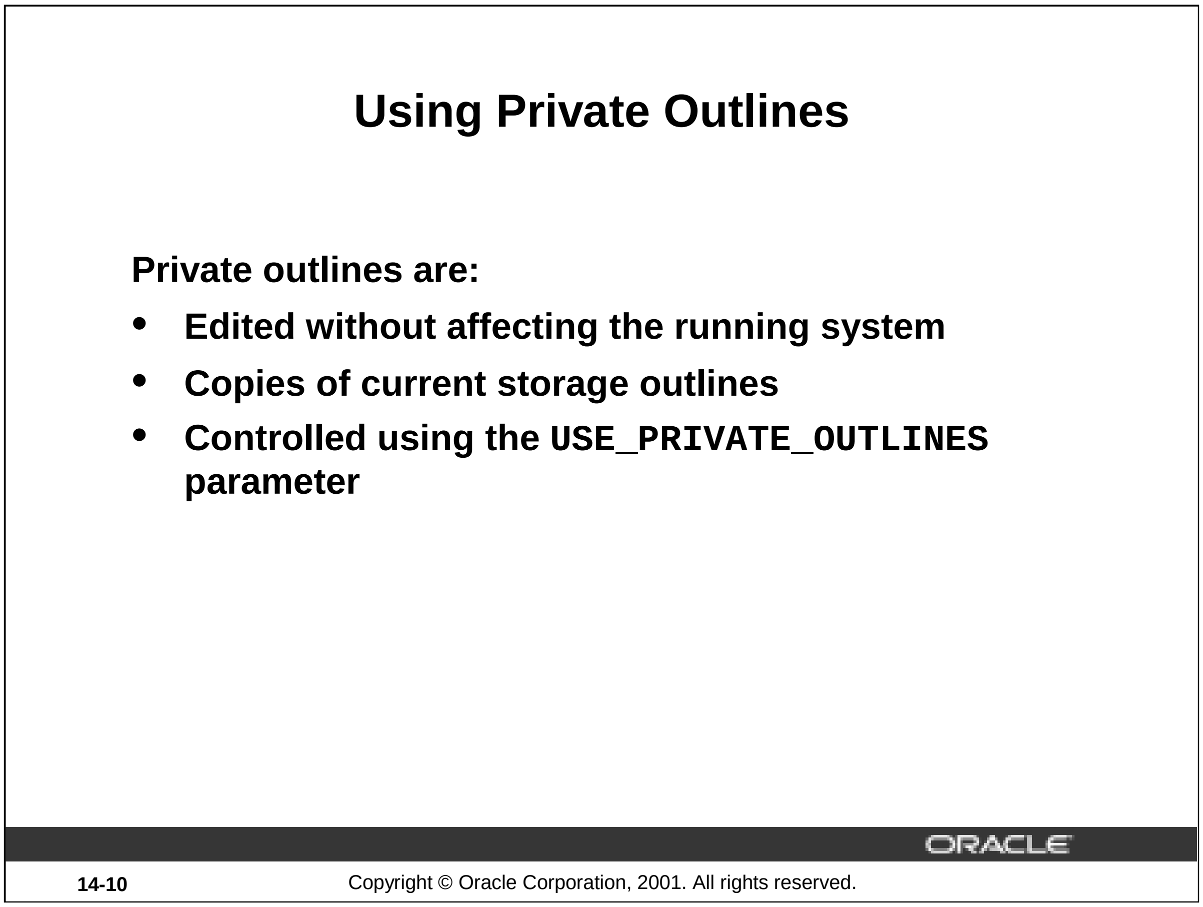
If `USE_STORED_OUTLINES` is set to TRUE, then outlines from the DEFAULT category are used. If `USE_STORED_OUTLINES` is set to a category name, then the outlines from that category are used. If there is no matching outline in that category, but there is one in the DEFAULT category, then that outline is used.

The statement must match the text of the statement in the outline. They are compared using the method for comparing cursors in the shared pool. This means that hints in the outline have to be used in the statement text to cause a match. Values in bind variables do not need to match.

To determine a SQL statement's execution plan, Oracle9*i* uses the following logic:

- The statement is compared to statements in the shared pool for matching text and outline category.
- If no matching statement is found, the data dictionary is queried for a matching outline.
- If a matching outline is found, Oracle9*i* integrates the outline into the statement and creates the execution plan.
- If no outline is found, the statement is executed using normal (non outline) methods.

If an outline specifies the use of an object that cannot be used (for example, it references an index that no longer exists), the statement will simply not use the hint. To verify that a stored outline is being used, the explain plan for a statement needs to be compared when running with and without `USE_STORED_OUTLINES` set.



Using Private Outlines

Private outlines are:

- **Edited without affecting the running system**
- **Copies of current storage outlines**
- **Controlled using the `USE_PRIVATE_OUTLINES` parameter**

Using Private Outlines

The `USE_PRIVATE_OUTLINES` parameter lets you control the use of private outlines. A private outline is an outline seen only in the current session and whose data resides in the current parsing schema. Any changes made to such an outline are not seen by any other session on the system, and applying a private outline to the compilation of a statement can only be done in the current session with the `USE_PRIVATE_OUTLINES` parameter. Only when you explicitly choose to save your edits back to the public area are they seen by the rest of the users. The outline is cloned into the user's schema at the onset of the outline editing session. All subsequent editing operations are performed on that clone until the user is satisfied with the edits and chooses to publicize them. In this way, any editing done by the user does not impact the rest of the user community, which would continue to use the public version of the outline until the edits are explicitly saved. When a private outline is created, an error is returned if the prerequisite outline tables to hold the outline data do not exist in the local schema. These tables can be created using the `DBMS_OUTLN_EDIT.CREATE_EDIT_TABLES` procedure. You can also use the `UTLEDITOL.SQL` script.

Prerequisites to Using Private Outlines

When the `USE_PRIVATE_OUTLINES` parameter is enabled and an outlined SQL statement is issued, the optimizer retrieves the outline from the session private area rather than the public area used when `USE_STORED_OUTLINES` is enabled. If no outline exists in the session private area, then the optimizer will not use an outline to compile the statement.

Editing Stored Outlines

Editing and using private outlines:

- **Create the outline tables in the current schema**
- **Copy the selected outline to private outline**
- **Edit the outline stored as a private outline**
- **To use the private outline, set the parameter `USE_PRIVATE_OUTLINE`**
- **To allow public access to the new stored outline, overwrite the stored outline**
- **Reset `USE_PRIVATE_OUTLINE` to `FALSE`**

ORACLE

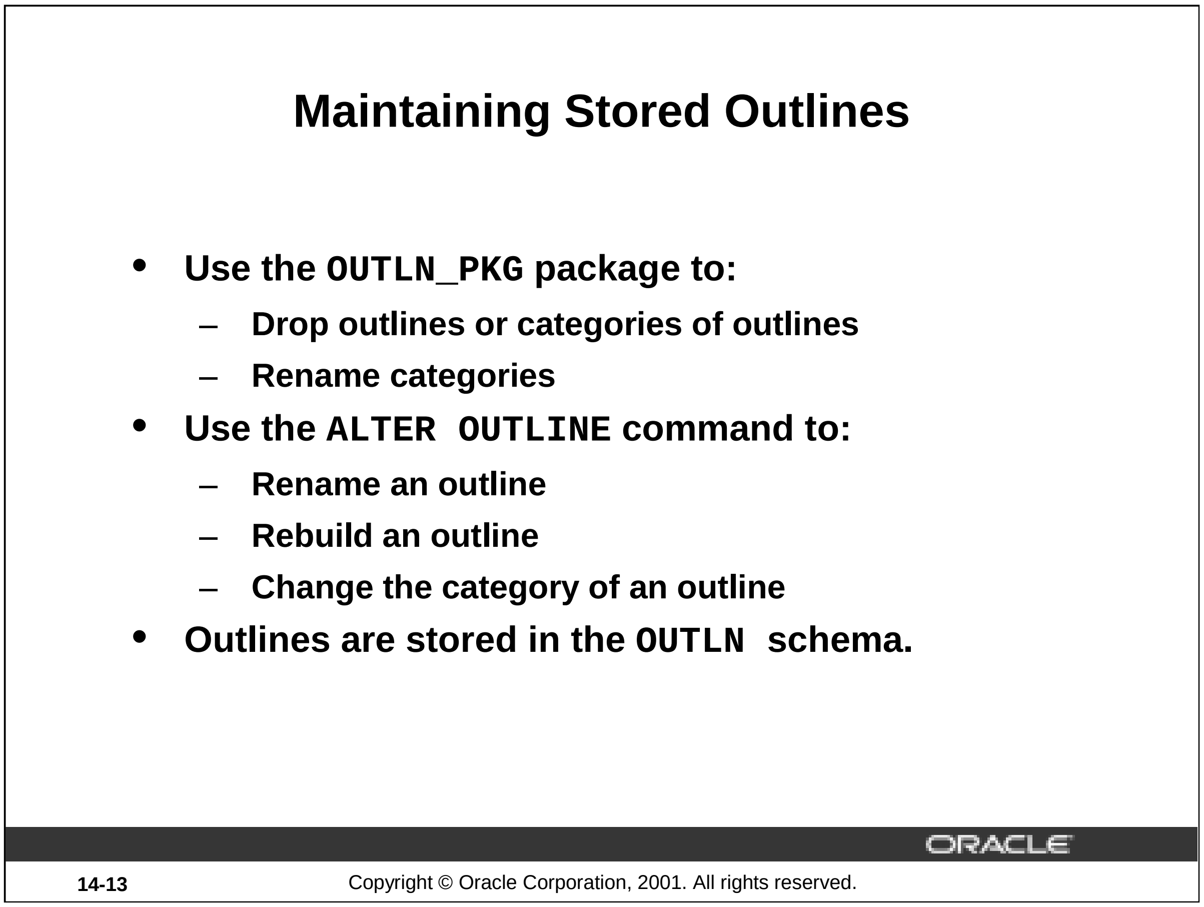
Editing Stored Outlines

Assume that you want to edit the outline DEV01. The steps are as follows:

1. Connect to a schema from which the outlined statement can be executed, and make sure that the `CREATE ANY OUTLINE` and `SELECT` privileges have been granted. Create outline editing tables locally with the `DBMS_OUTLN_EDIT.CREATE_EDIT_TABLES` procedure.
2. Clone the outline being edited to the private area using the following:
   ```
   CREATE PRIVATE OUTLINE p_DEV01 FROM dev01;
   ```
3. Edit the outline, either with the Outline Editor in Enterprise Manager or manually by querying the local `OL$HINTS` tables and performing DML against the appropriate hint tables. To change the join order, use the procedure `DBMS_OUTLN_EDIT.CHANGE_JOIN_POS`
4. If manually editing the outline, then re-synchronize the stored outline definition using the following so-called identity statement:
   ```
   CREATE PRIVATE OUTLINE p_dev01 FROM PRIVATE p_dev01;
   ```

Editing Stored Outlines (continued)

5. You can also use `DBMS_OUTLN_EDIT.REFRESH_PRIVATE_OUTLINE or ALTER SYSTEM FLUSH SHARED_POOL` to accomplish the same task as in step 4.
6. Test the edits. Set `USE_PRIVATE_OUTLINES=TRUE`, and issue the outline statement or run `EXPLAIN PLAN` on the statement.
7. If you want to preserve these edits for public use, then publicize the edits with the following statement. `CREATE OR REPLACE OUTLINE dev01 FROM PRIVATE p_dev01;`
8. Disable private outline usage by setting the parameter `USE_PRIVATE_OUTLINES=false`



Maintaining Stored Outlines

- Use the `OUTLN_PKG` package to:
  – Drop outlines or categories of outlines
  – Rename categories
- Use the `ALTER OUTLINE` command to:
  – Rename an outline
  – Rebuild an outline
  – Change the category of an outline
- Outlines are stored in the `OUTLN` schema.

14-13

Maintaining Stored Outlines

Use procedures in the `OUTLN_PKG` package to manage stored outlines and their categories. These procedures are:

- `Drop_unused`: Drops outlines that have not been used since they were created
- `Drop_by_cat`: Drops outlines assigned to the specific category name
- `Update_by_cat`: Reassigns outlines from one category to another

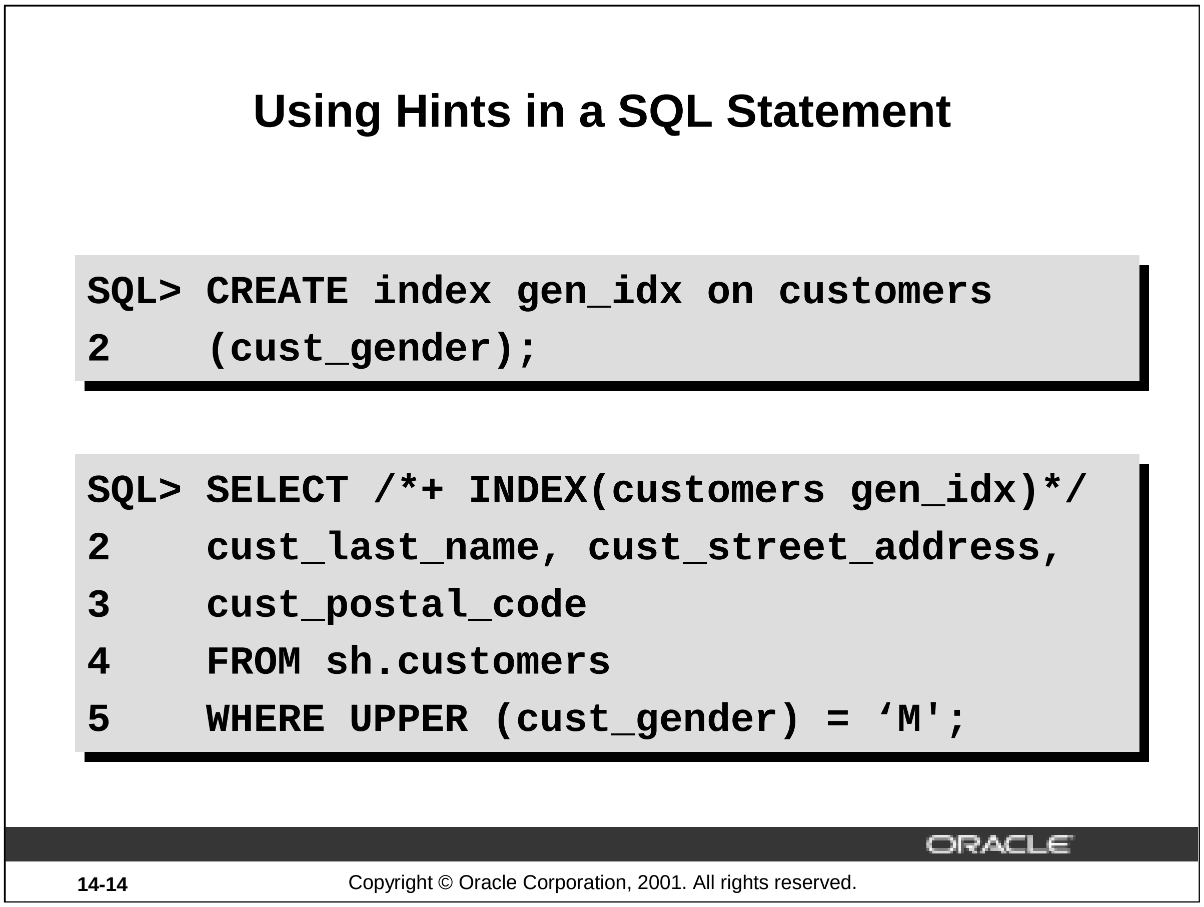
Outlines can also be managed with the `ALTER/DROP OUTLINE` commands.

Plans may be exported and imported by exporting the `OUTLN` schema, where all outlines are stored. Outlines can be queried from tables in the schema:

- `OL$`: Outline name, category, creation timestamp, and the text of the statement
- `OL$HINTS`: The hints for the outlines in `OL$`

The equivalent data dictionary views are: `DBA_OUTLINES` and `DBA_OUTLINE_HINTS`.

Note: Because the user `OUTLN` is automatically created with the database, its password should be changed.



Using Hints in a SQL Statement

```
SQL> CREATE index gen_idx on customers
2    (cust_gender);
```

```
SQL> SELECT /*+ INDEX(customers gen_idx)*/
2    cust_last_name, cust_street_address,
3    cust_postal_code
4    FROM sh.customers
5    WHERE UPPER (cust_gender) = ‘M';
```

14-14

Using Hints in a SQL Statement

You may know information about your data that the optimizer does not know. For example, you may know that a certain index is more selective for certain queries. Based on this information, you may be able to choose a more efficient execution plan than the optimizer. In such a case, use hints to force the optimizer to use the optimal execution plan.

Example

In the above example there is an index on the `Cust_gender` column of the customers table. There are very few male customers, therefore the statement runs faster using the `gen_idx` index. In order to force the optimizer to use the index, it is put in a HINT.

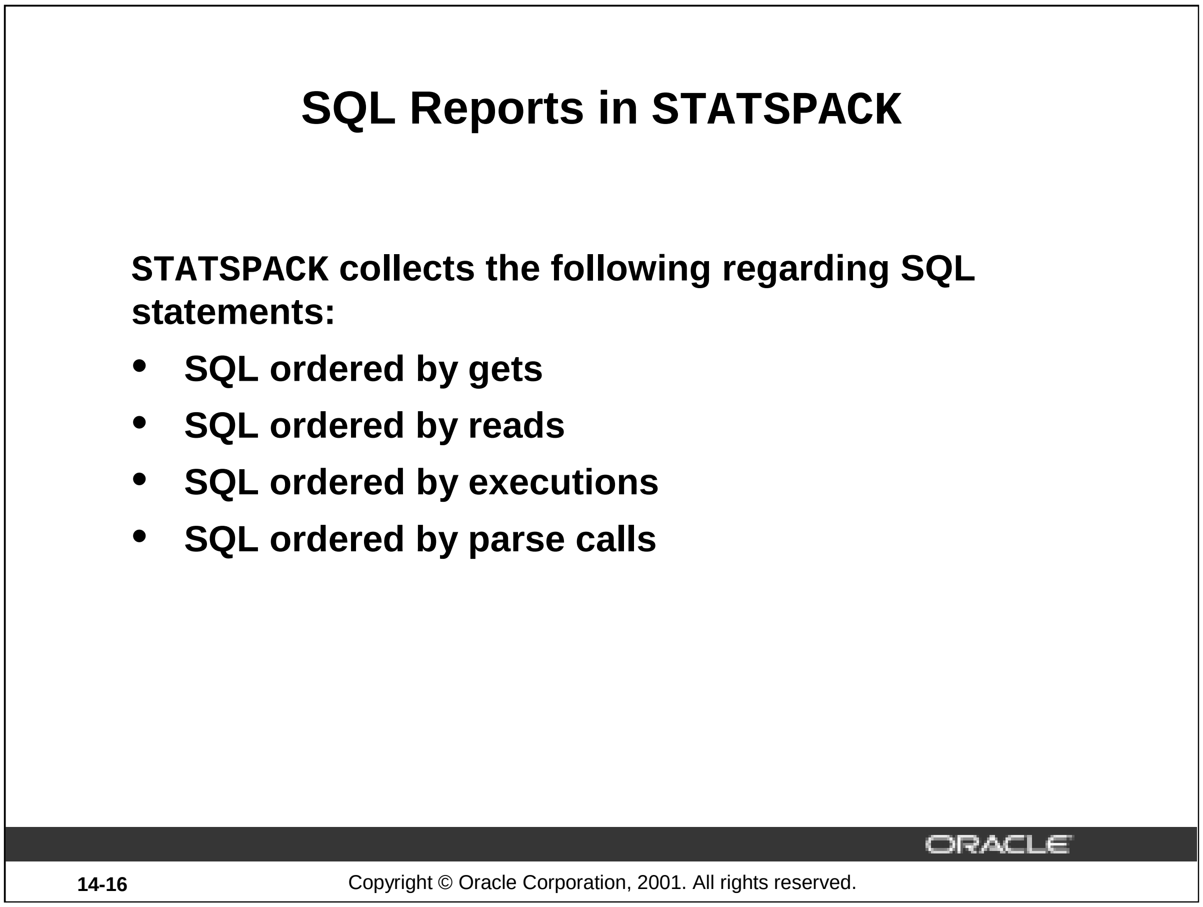
Overview of Diagnostic Tools

- `STATSPACK`
- `EXPLAIN PLAN`
- SQL trace and `TKPROF`
- SQL*Plus autotrace feature
- Oracle SQL Analyze

Overview of Diagnostic Tools

Numerous diagnostic tools are available for evaluating the performance of SQL statements and PL/SQL modules. Each provides a developer or DBA with a varying degree of information:

- `STATSPACK`: This utility collects information regarding database statistics and SQL statements.
- `EXPLAIN PLAN`: This is executed within a session for a SQL statement.
- `SQL Trace`: This utility provides detailed information regarding the execution of SQL statements.
- `TKPROF`: This is an operating system utility that takes the output from a SQL TRACE session and formats it into a readable format.
- Autotrace: This is a SQL*Plus feature. Autotrace generates an execution plan for a SQL statement and provides statistics relative to the processing of that statement.
- Oracle SQL Analyze: This is part of the Oracle Enterprise Manager Tuning Pack, and it provides a powerful user interface for tuning SQL statements.



SQL Reports in STATSPACK

STATSPACK collects the following regarding SQL statements:

- **SQL ordered by gets**
- **SQL ordered by reads**
- **SQL ordered by executions**
- **SQL ordered by parse calls**

14-16

SQL Reports in STATSPACK

STATSPACK gives four different views based upon SQL statements stored in the SHARED POOL at the time of either the beginning snapshot, or the ending snapshot. The report provides these statements in four different sections. These sections are:

- SQL ordered by gets
- SQL ordered by reads
- SQL ordered by executions
- SQL ordered by parse calls

These views can be examined to see which SQL statements are having the most impact on the performance of the database. These are the best SQL statements to tune because tuning them is most likely to improve performance dramatically.

Explain Plan

- **Can be used without tracing**
- **Needs the `PLAN_TABLE` table:**

```
SQL> @$ORACLE_HOME/rdbms/admin/utlxplan
```

- **Create the explain plan:**

```
SQL> Explain plan for
  2  select last_name from hr.employees;
```

- **Query `plan_table` to display the execution plans:**
  – **Query `PLAN_TABLE` directly**
  – **Use script utlxpls.sql (Hide Parallel Query information)**
  – **Use script utlxplp.sql (Show parallel Query information)**

The Explain Plan Statement

You can use the EXPLAIN PLAN statement in SQL*Plus without using tracing. You need to create a table called `plan_table` using the supplied `utlxplan.sql` script. The useful columns for most purposes are operation, options, and object_name.

To explain the plan for a query, use the following syntax:

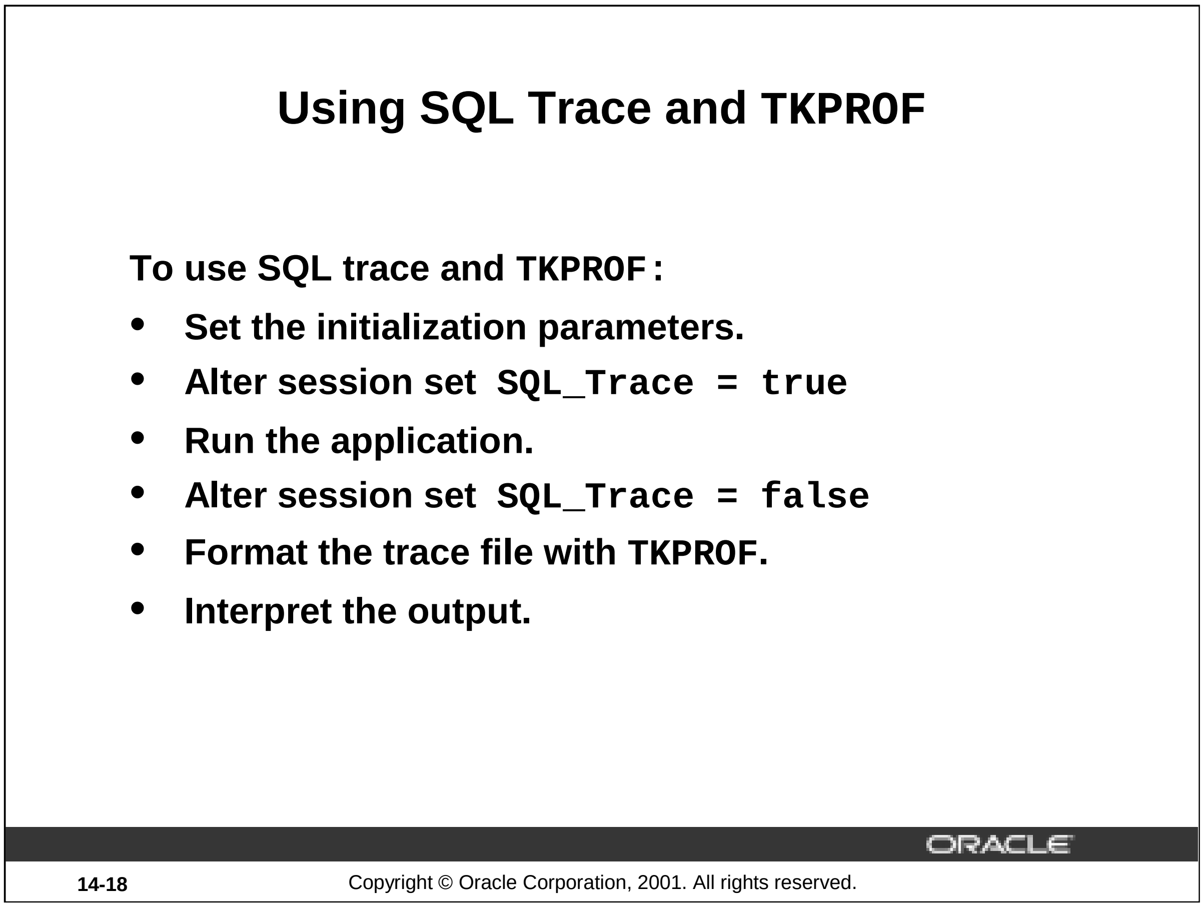
```
EXPLAIN PLAN [SET STATEMENT_ID = '...'] [INTO
my_plan_table]
FOR SELECT ….
```

Then query plan_table to check the execution plan. Plan_table shows you how the statement would be executed if you chose to run it at that moment. Remember that if you make changes before running the statement (creating an index, for example), the actual execution may be different. Also, if you do not use a `statement_id` in the EXPLAIN PLAN statement, you may want to truncate the plan_table prior to generating another execution plan.

Querying `PLAN_TABLE`

Plan_table can be queried directly, or you can run either `utlxpls.sql` or `utlxplp.sql` (depending on whether Parallel Query statistics are required. These scripts show the most commonly selected columns of plan_table.

Note: For additional information on the columns in the `plan_table`, see the *Oracle9i Performance Guide and Reference* manual.



Using SQL Trace and `TKPROF`

To use SQL trace and `TKPROF`:

- **Set the initialization parameters.**
- **Alter session set `SQL_Trace = true`**
- **Run the application.**
- **Alter session set `SQL_Trace = false`**
- **Format the trace file with `TKPROF`.**
- **Interpret the output.**

Using SQL Trace and TKPROF

A particular sequence of steps is necessary to properly diagnose SQL statement performance with SQL Trace and TKPROF:

- The first step is to ensure appropriate initialization parameters. These can be set at the instance level; some parameters may also be set at the session level.
- SQL Trace must be invoked at either the instance or session level. Generally, it is better if it is invoked at the session level.
- Run the application or SQL statement you want to diagnose.
- Turn off SQL Trace. This is necessary to properly close the trace file at the operating system level.
- Use TKPROF to format the trace file generated during the trace session. Unless the output file is formatted, it is very difficult to interpret the results.
- Use the output from `TKPROF` to diagnose the performance of the SQL statement.

Initialization Parameters

Two parameters in the `init.ora` file control the size and destination of the output file from the SQL trace facility:

```
max_dump_file_size = n
```

This parameter is measured in bytes if K or M is specified, otherwise the number represents operating system blocks. The default value is 10,000 operating system blocks.

When a trace file exceeds the size defined by the parameter value, the following message appears at the end of the file: `*** Trace file full ***`

The following parameter determines the trace file destination:

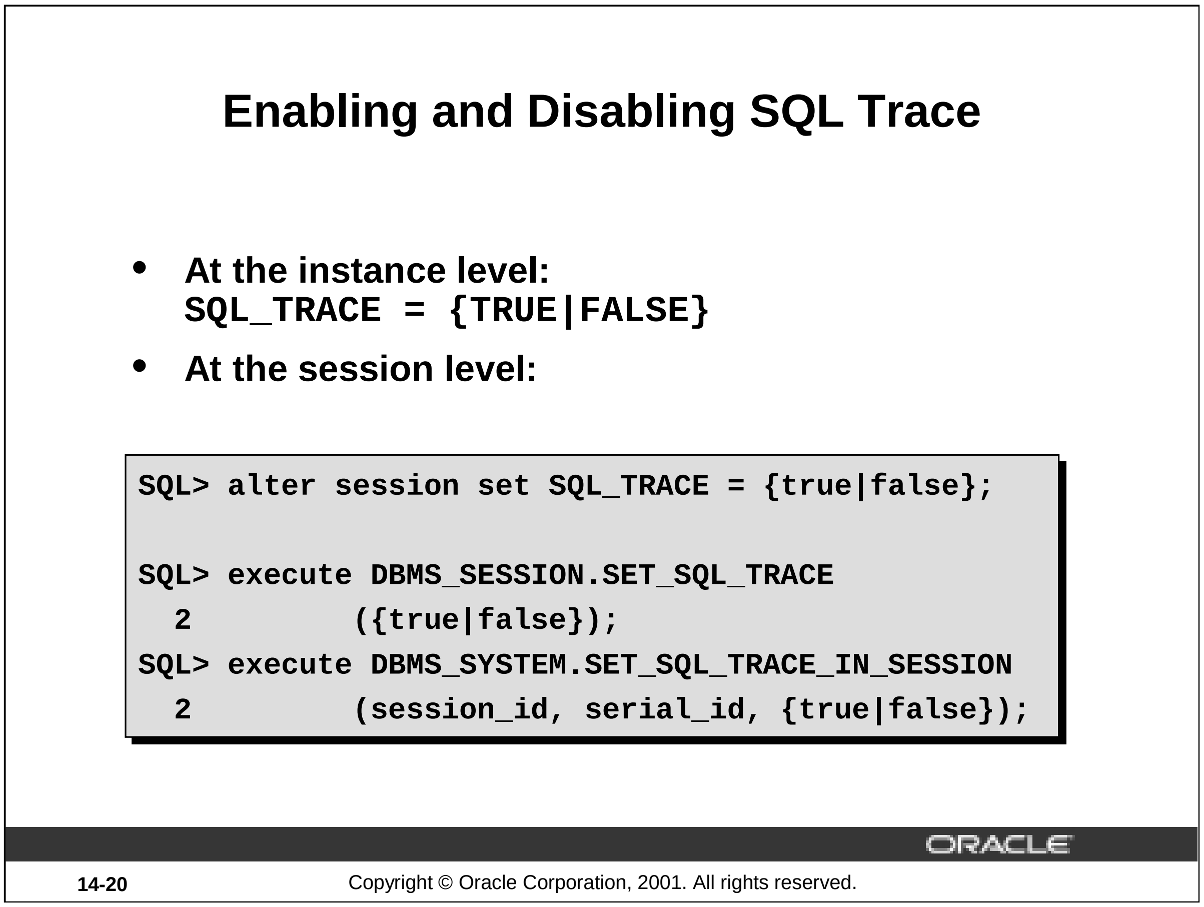
```
user_dump_dest = directory
```

You must set a third parameter to get timing information:

```
timed_statistics = TRUE
```

The timing statistics have a resolution of one one-hundredth of a second.

The `TIMED_STATISTICS` parameter can also be set dynamically at the session level, using the ALTER SESSION command.



Enabling and Disabling SQL Trace

- **At the instance level:**
  `SQL_TRACE = {TRUE|FALSE}`
- **At the session level:**

```
SQL> alter session set SQL_TRACE = {true|false};

SQL> execute DBMS_SESSION.SET_SQL_TRACE
  2         ({true|false});
SQL> execute DBMS_SYSTEM.SET_SQL_TRACE_IN_SESSION
  2         (session_id, serial_id, {true|false});
```

14-20

Enabling and Disabling SQL Trace

SQL Trace can be enabled or disabled using different methods at either the instance or the session level.

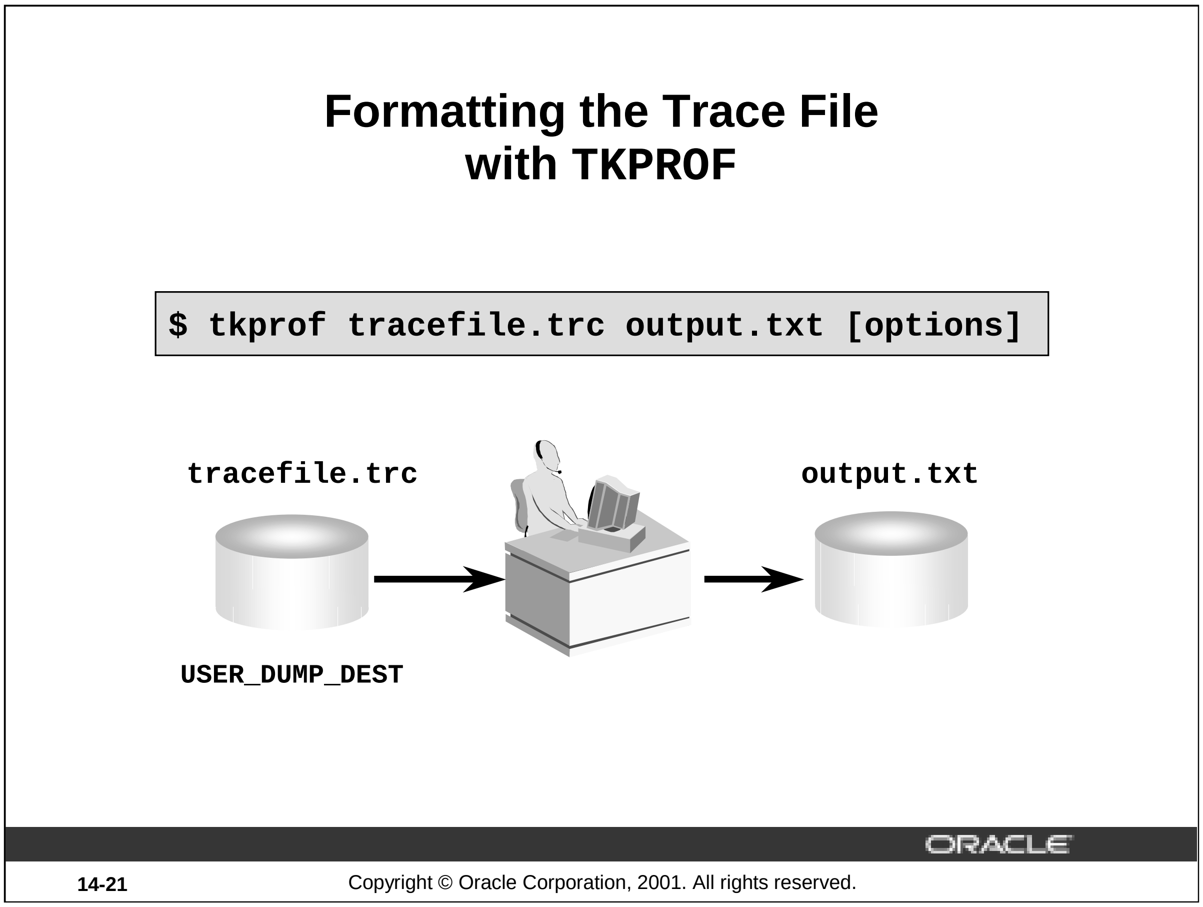
Instance Level

Setting the `SQL_TRACE` parameter at the instance level is one way to enable tracing. However, it requires that the instance be shut down, then restarted when tracing is no longer needed. This method also imposes a significant performance cost, because all sessions for the instance are traced.

Session Level

Session-level tracing results in less of a cost in overall performance, because specific sessions can be traced. Three methods for enabling or disabling SQL Trace are:

- Using the ALTER SESSION command, which results in tracing for the duration of the session or until the value is set to FALSE
- Using the `DBMS_SESSION.SET_SQL_TRACE` procedure for the session
- Using the `DBMS_SYSTEM.SET_SQL_TRACE_IN_SESSION` procedure to enable tracing for a user other than the logged-in user at the session level



Formatting the Trace File with TKPROF

```
$ tkprof tracefile.trc output.txt [options]
```

14-21 Copyright © Oracle Corporation, 2001. All rights reserved.

Formatting the Trace File with TKPROF

Use TKPROF to format the trace file into a readable output:

```
tkprof tracefile outputfile [sort=option] [print=n]
[explain=username/password] [insert=filename] [sys=NO]
[record=filename] [table=schema.tablename]
```

The trace file is created in the directory specified by the `USER_DUMP_DEST` parameter and the output is placed in the directory specified by the output file name.

SQL Trace also gathers statistics for recursive SQL statements. You cannot directly affect the amount of recursive SQL that the server executes, so these figures are very useful in themselves. Use the SYS=NO option of TKPROF to suppress the output of these figures.

When you specify the EXPLAIN parameter, TKPROF logs on to the database with the username and password provided. It then works out the access path for each SQL statement traced and includes that in the output. Because TKPROF logs on to the database, it uses the information available at the time that TKPROF is run, not at the time the trace statistics were produced. This could make a difference if, for example, an index has been created or dropped since tracing the statement.

TKPROF also reports library cache misses: This indicates the number of times that the statement was not found in the library cache.

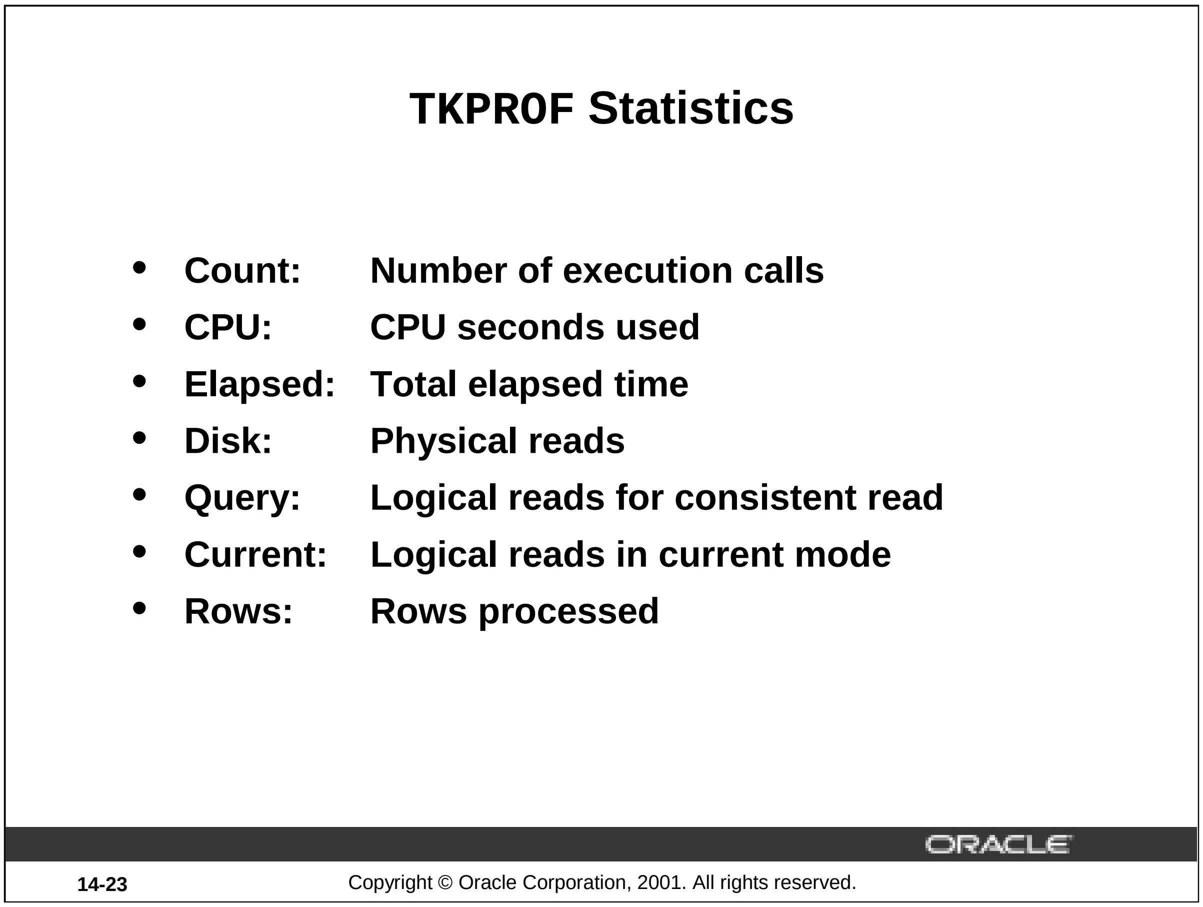
TKPROF Options

Option	**Description**
TRACEFILE	The name of the trace output file
OUTPUTFILE	The name of the formatted file
SORT=*option*	The order in which to sort the statements
PRINT=*n*	Print the first *n* statements
EXPLAIN=*user/password*	Run EXPLAIN PLAN in the specified username
INSERT=*filename*	Generate INSERT statements
SYS=NO	Ignore recursive SQL statements run as user `sys`
AGGREGATE=[Y/N]	If you specify AGGREGATE = NO, TKPROF does not aggregate multiple users of the same SQL text
RECORD=*filename*	Record statements found in the trace file
TABLE=*scheme.tablename*	Put execution plan into specified table (rather than the default of plan_table)

You can enter tkprof at the operating system to get a listing of all the available options and output.

Note: The sort options are the following:

Sort Option	**Description**
Prscnt, execnt, fchcnt	Number of times parse, execute, and fetch were called
prscpu, execpu, fchcpu	CPU time parsing, executing, and fetching
prsela, exela, fchela	Elapsed time parsing, executing, and fetching
prsdsk, exedsk, fchdsk	Number of disk reads during parse, execute, and fetch
prsqry, exeqry, fchqry	Number of buffers for consistent read during parse, execute, fetch
prscu, execu, fchcu	Number of buffers for current read during parse, execute, fetch
prsmis, exemis	Number of misses in library cache during parse, and execute
exerow, fchrow	Number of rows processed during execute, and fetch
userid	User ID of user who parsed the cursor



TKPROF Statistics

• Count:	Number of execution calls
• CPU:	CPU seconds used
• Elapsed:	Total elapsed time
• Disk:	Physical reads
• Query:	Logical reads for consistent read
• Current:	Logical reads in current mode
• Rows:	Rows processed

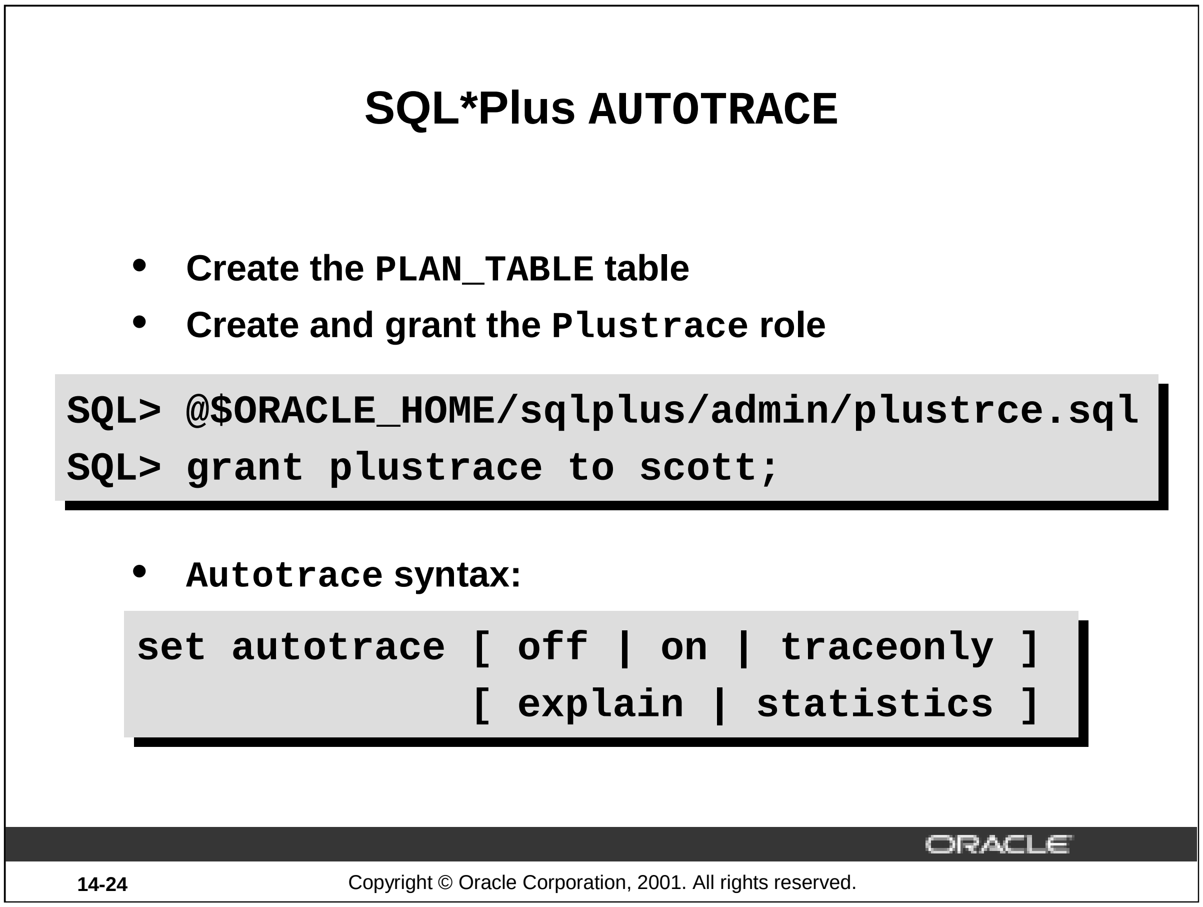
14-23

TKPROF Statistics

TKPROF will collect the following statistics:

- Count - The number of times the statement was parsed or executed and the number of fetch calls issued for the statement
- CPU - Processing time for each phase, in seconds (if the statement was found in the shared pool, this is 0 for the parse phase)
- Elapsed - Elapsed time, in seconds (this is not usually very helpful, because other processes affect elapsed time)
- Disk - Physical data blocks read from the database files (usually the statistic is quite low if the data was buffered)
- Query - Logical buffers retrieved for consistent read (usually for SELECT statements)
- Current - Logical buffers retrieved in current mode (usually for DML statements)
- Rows - Rows processed by the outer statement (for SELECT statements, this is shown for the fetch phase; for DML statements, it is shown for the execute phase)

The sum of Query and Current is the total number of logical buffers accessed.



SQL*Plus AUTOTRACE

- Create the PLAN_TABLE table
- Create and grant the Plustrace role

```
SQL> @$ORACLE_HOME/sqlplus/admin/plustrce.sql
SQL> grant plustrace to scott;
```

- Autotrace syntax:

```
set autotrace [ off | on | traceonly ]
              [ explain | statistics ]
```

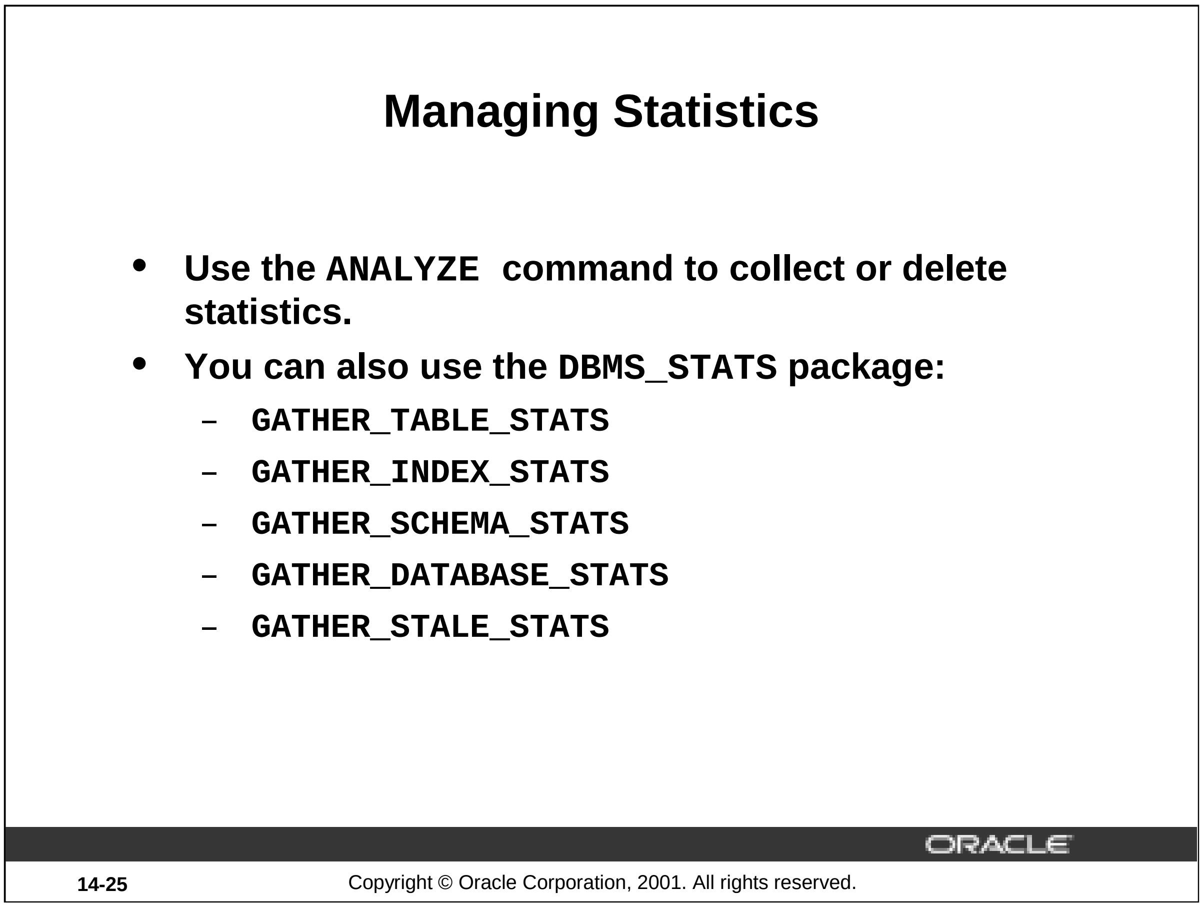
SQL*Plus AUTOTRACE

SQL*Plus AUTOTRACE can be used instead of SQL Trace. The advantage in using Autotrace is that you do not have to format a trace file and it automatically displays the execution plan for the SQL statement.

However, Autotrace does parse and execute the statement, whereas explain-plan only parses the statement.

The steps for using Autotrace are:

1. Create the `Plan_table` table using the `utlxplan.sql` script.
2. Create the Plustrace role by executing the `plustrce.sql` script. This grants SELECT privileges on V$ views to the role and grants the role to the DBA role. Grant the Plustrace role to users who do not have the DBA role.
3. Use Autotrace.
   – ON: Produces the result set and the explain plan, and lists statistics
   – TRACEONLY: Displays the explain plan and the statistics; you will not see the result set, although the statement is executed
   – ON EXPLAIN: Displays the result set and the explain-plan results, without the statistics
   – TRACEONLY STATISTICS: Displays the statistics only



Managing Statistics

- Use the `ANALYZE` command to collect or delete statistics.
- You can also use the `DBMS_STATS` package:
  - `GATHER_TABLE_STATS`
  - `GATHER_INDEX_STATS`
  - `GATHER_SCHEMA_STATS`
  - `GATHER_DATABASE_STATS`
  - `GATHER_STALE_STATS`

Managing Statistics

You collect statistics on an object with the `ANALYZE` command. Although the cost-based optimizer is not sensitive to minor changes in volume or selectivity, you may want to collect new statistics periodically on frequently modified tables to ensure that the optimizer is using recent, accurate information.

```
ANALYZE {INDEX|TABLE|CLUSTER} object_name
{COMPUTE|DELETE|ESTIMATE} STATISTICS
[FOR ... [SIZE n]] [SAMPLE n {ROWS|PERCENT}]
```

Using the `ANALYZE` command to collect new statistics overwrites any existing statistics in the data dictionary and flushes any related execution plans from the shared pool.

The `DBMS_STATS` package contains several procedures that allows an index, table, schema or database to be analyzed. This package also enables you to gather most of the statistics with a degree of parallelism. For detailed information, refer to the *Oracle9i Supplied Packages Reference* manual.

Statistics Accuracy

`COMPUTE`: Calculates exact statistics. It performs a full table scan and several calculations. For large tables this operation may take a long time.

`ESTIMATE`: You estimate statistics with this option. If you use this option with a suitable sample of the data, it is almost as accurate as the `COMPUTE` option.

Statistics Accuracy (continued)

`DELETE`: You clear out statistics with this option. You do not need to use this option before reanalyzing an object, because existing statistics are overwritten.

The `FOR` Clause

The `FOR` clause offers the following options:

- `FOR TABLE`: Restricts the statistics collected to table statistics only rather than table and column statistics
- `FOR COLUMNS`: Restricts the statistics collected to only column statistics for the specified columns, rather than for all columns and attributes
- `FOR ALL COLUMNS`: Collects column statistics for all columns.
- `FOR ALL INDEXED COLUMNS`: Collects column statistics for all indexed columns in the table
- `FOR ALL [LOCAL] INDEXES`: Specifies that all indexes associated with the table will be analyzed. `LOCAL` specifies that all local index partitions are analyzed.

The `SIZE` Clause

The `SIZE` clause specifies the maximum number of histogram buckets. The default value is 75, and the maximum value is 254. Histograms are discussed in more detail later in this lesson.

Table Statistics

- **Number of rows**
- **Number of blocks and empty blocks**
- **Average available free space**
- **Number of chained or migrated rows**
- **Average row length**
- **Last `ANALYZE` date and sample size**
- **Data dictionary view: `DBA_TABLES`**

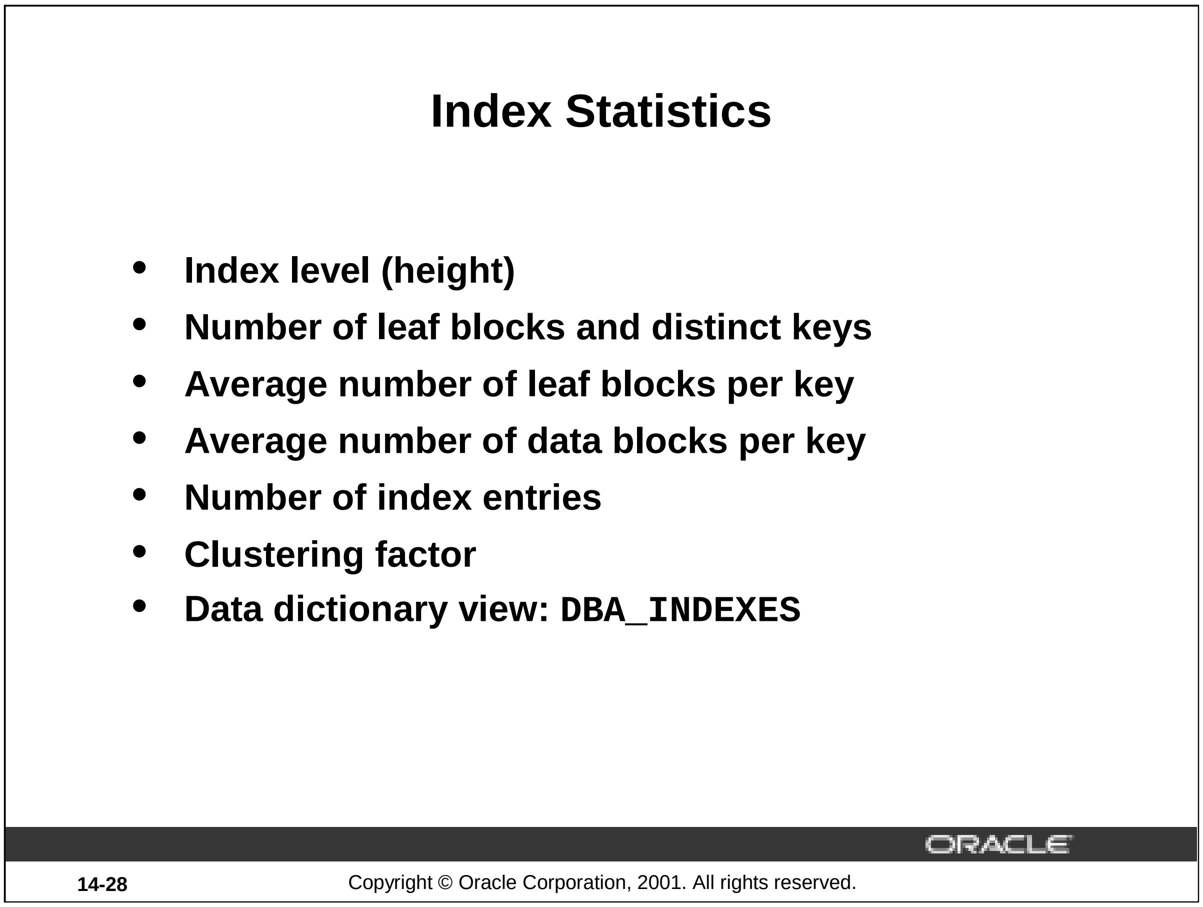
Number of Blocks and Empty Blocks

Each table maintains a high-water mark in the segment header block. The high-water mark indicates the last block that was ever used for the table. When the Oracle server performs full table scans, it reads all the blocks up to the high-water mark. Note that the high-water mark is not reset when rows are deleted from the table.

The `DBMS_SPACE.UNUSED_SPACE` procedure can also be used to find the high-water mark and the number of blocks above the high-water mark, if analyzing a table is impossible or undesirable.

Example

```
SQL> analyze table employees estimate statistics for table;
SQL> select num_rows,blocks,empty_blocks
  2  ,      avg_space,avg_row_len, sample_size
  3 from   dba_tables
  4 where  table_name = ‘ORDERS' and owner = ‘SH’;
NUM_ROW SBLOCKS EMPTY_BLOCKS AVG_SPACE AVG_ROW_LEN SAMPLE_SIZE
------- ------- ------------ --------- ----------- -----------
  15132     434            5       225          48        1064
```



Index Statistics

- Index level (height)
- Number of leaf blocks and distinct keys
- Average number of leaf blocks per key
- Average number of data blocks per key
- Number of index entries
- Clustering factor
- Data dictionary view: `DBA_INDEXES`

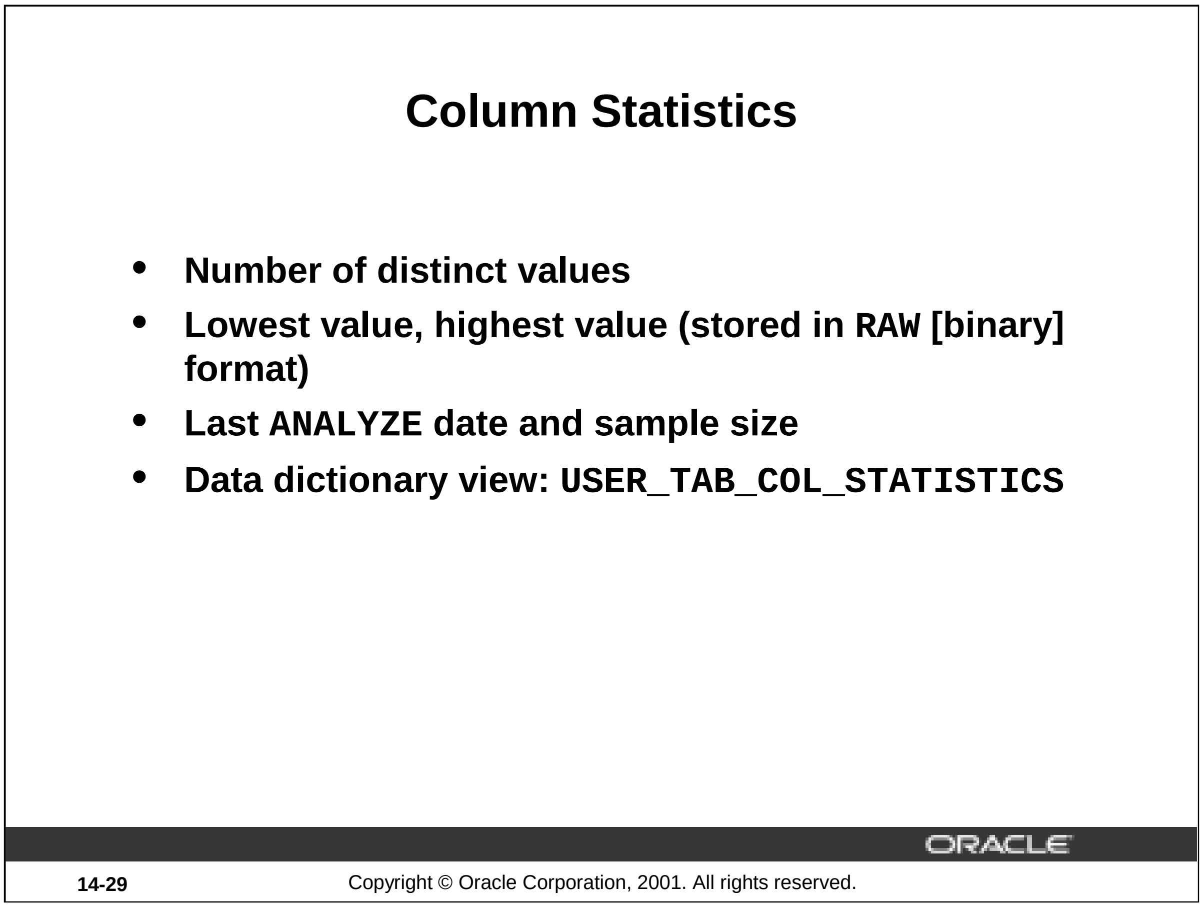
14-28

Clustering Factor

The index clustering factor is an important index statistic for the cost-based optimizer to estimate index scan costs. It is an indication of the number of (logical) data block visits needed to retrieve all table rows by means of the index. If the index entries follow the table row order, this value approaches the number of data blocks (each block is visited only once); on the other hand, if the index entries randomly point at different data blocks, the clustering factor could approach the number of rows.

Example

```
SQL> analyze index reg_pk compute statistics;
SQL>select blevel, leaf_blocks, distinct_keys,
clustering_factor
  2  from   dba_indexes
  3  where  index_name = ’REG_PK’ and owner = user;
   BLEVEL  LEAF_BLOCKS   DISTINCT_KEYS     CLUSTERING_FACTOR
 --------  -----------   -------------     -----------------
        2          682           56252                 21349
```



Column Statistics

- **Number of distinct values**
- **Lowest value, highest value (stored in `RAW` [binary] format)**
- **Last `ANALYZE` date and sample size**
- **Data dictionary view: `USER_TAB_COL_STATISTICS`**

14-29

The `USER_TAB_COL_STATISTICS` view

The USER_TAB_COL_STATISTICS view offers a relevant subset of the columns displayed by the `USER_TAB_COLUMNS` view. You can also use the `DBA_TAB_COL_STATISTICS` view; note however that this view does not contain an owner column, so you get confusing results when your database contains multiple tables with the same name in different schemas.

The `Num_buckets` column shows that regular column statistics are treated as a histogram with one bucket. Histograms are discussed on the next page.

Example

```
SQL> select column_name, num_distinct, low_value, high_value
  2 ,     num_nulls, num_buckets
  3 from  user_tab_col_statistics
  4 where table_name = 'EMPLOYEES' and column_name = 'SALARY';
 COLUMN_NAME  NUM_DIST LOW_VALUE  HIGH_VALUE  NUM_NULLS NUM_BUCKETS
 -----------  -------- ---------  ----------  --------- -----------
      SALARY       496      C208      C30A62          0           1
```

Histograms

- Histograms describe the data distribution of a particular column in more detail.
- They give better predicate selectivity estimates for unevenly distributed data.
- You create histograms with the `ANALYZE TABLE ... FOR COLUMNS ...` command
- Data dictionary views: `DBA_HISTOGRAMS`, `DBA_TAB_HISTOGRAMS`

ORACLE

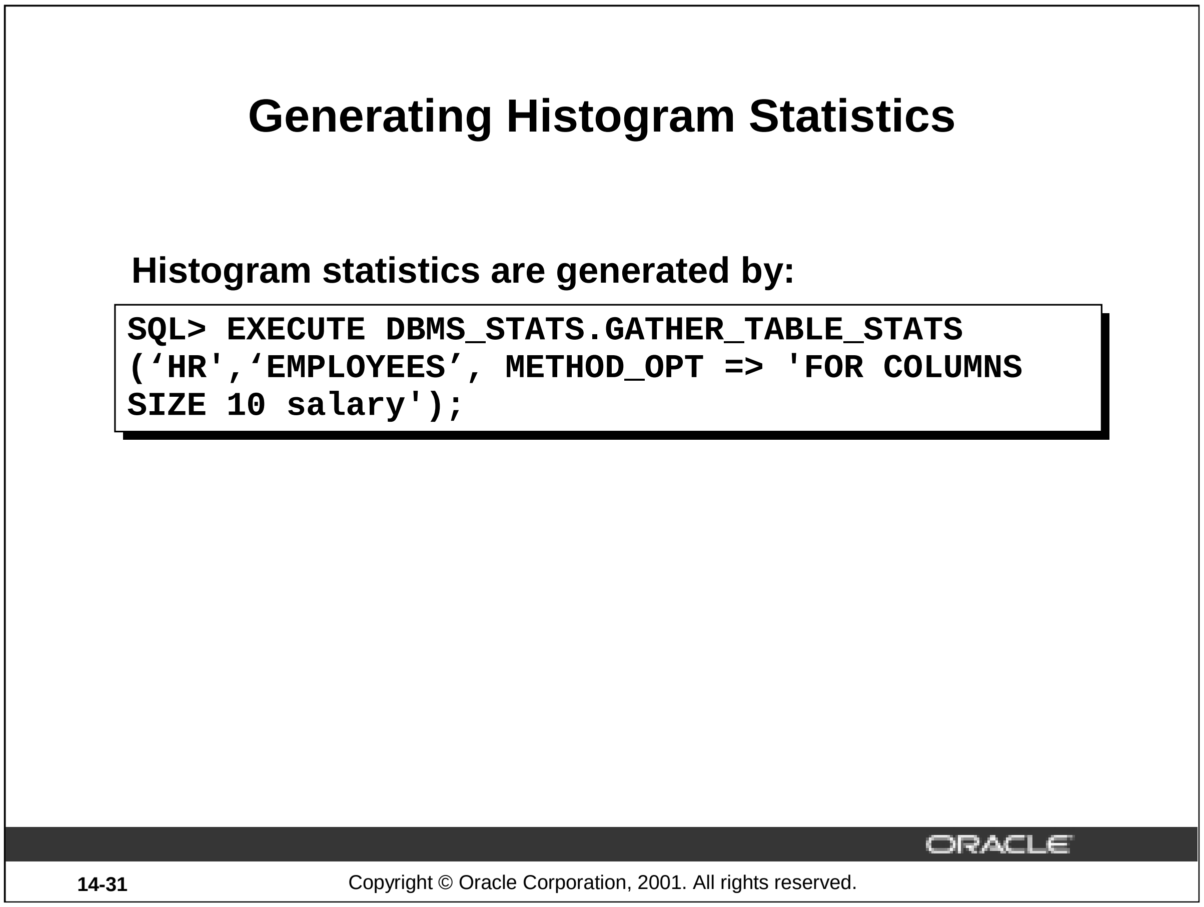
Histograms

When using regular column statistics, a minimum value and a maximum value are stored for each column. The cost-based optimizer uses these values to calculate predicate selectivity, assuming an even distribution of the data between those two extreme values. However, if your data is skewed, this assumption may lead to suboptimal plans. You can use histograms to store more detailed information about the data distribution within a column by partitioning the column values in a number of buckets. Note however that this means additional data dictionary storage requirements. Buckets are height balanced, meaning that each bucket contains approximately the same number of values.

The default number of buckets is 75, and the maximum value is 254.

Example

```
SQL>analyze table orders compute statistics for columns
SALES_REP_ID;
SQL>select endpoint_number, endpoint_value
    2  from    user_tab_histograms
    3  where  table_name = ‘ORDERS' and column_name =
'SALES_REP_ID';
ENDPOINT        ENDPOINT
_NUMBER           _VALUE
-------               ------
 1469                   156
 3939                   158
```



Generating Histogram Statistics

Histogram statistics are generated by:

```
SQL> EXECUTE DBMS_STATS.GATHER_TABLE_STATS
(‘HR',‘EMPLOYEES’, METHOD_OPT => 'FOR COLUMNS
SIZE 10 salary');
```

14-31

Generate Histogram Statistics

You generate histograms by using the `DBMS_STATS` package. You can generate histograms for columns of a table or partition. For example, to create a 10-bucket histogram on the SAL column of the `emp` table, issue the following statement:

```
EXECUTE DBMS_STATS.GATHER_TABLE_STATS (‘HR',‘EMPLOYEES’,
  METHOD_OPT => 'FOR COLUMNS SIZE 10 salary');
```

The SIZE keyword declares the maximum number of buckets for the histogram.

Choosing the Number of Buckets for a Histogram

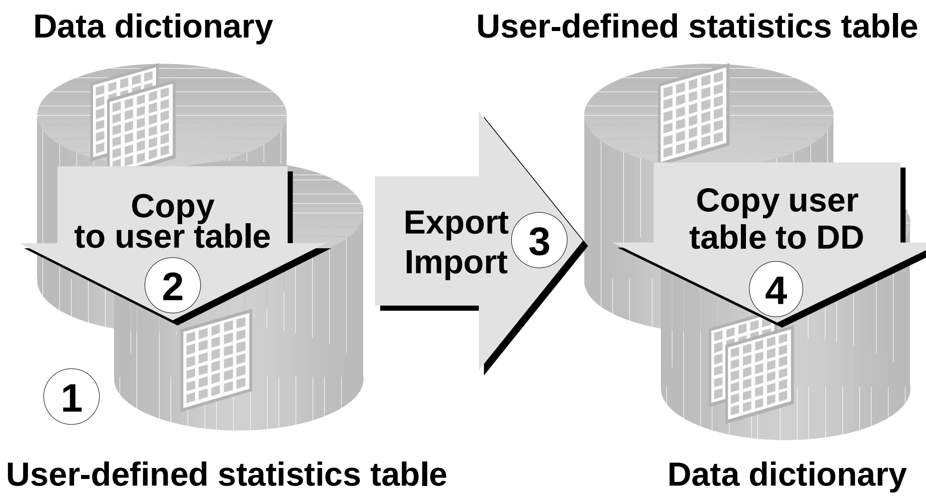
If the number of frequently occurring distinct values in a column is relatively small, then set the number of buckets to be greater than that number. The default number of buckets for a histogram is 75. This value provides an appropriate level of detail for most data distributions. However, because the number of buckets in the histogram, and the data distribution both affect a histogram’s usefulness, you might need to experiment with different numbers of buckets to obtain optimal results.

Generate Histogram Statistics (continued)

Example

```
SQL>analyze table orders compute statistics for columns order_id;
SQL>select endpoint_number, endpoint_value
    2  from    user_histograms
    3  where  table_name = ‘ORDERS' and column_name = ‘ORDER_ID';
ENDPOINT ENDPOINT
_NUMBER   _VALUE
-------   ------
0         50008
1         55198
2         56718
3         63037
...       ...
75        155801
```

Copying Statistics Between Databases



Copying Statistics Between Databases

By using the `DBMS_STATS` package procedures, you can copy statistics from an Oracle9*i* production to a test database to facilitate tuning. For example, to copy a schema's statistics:

1. Use the `DBMS_STATS.CREATE_STAT_TABLE` procedure in the production database to create a user-defined statistics table.
2. Use the `DBMS_STATS.EXPORT_SCHEMA_STATS` procedure in the production database to copy statistics from the data dictionary to the user-defined statistics table from step 1.
3. Use the Export and Import utilities to transfer the statistics to a corresponding user-defined statistics table in the test database.
4. Use the `DBMS_STATS.IMPORT_SCHEMA_STATS` procedure to import the statistics into the data dictionary in the test database.

The `DBMS_STATS` package can also be used to back up statistics prior to analyzing objects. The backup can used to:

- Restore old statistics
- Study changes in data characteristics over time

Example: Copying Statistics

Step 1. Create the table to hold the statistics:

```
DBMS_STATS.CREATE_STAT_TABLE
('TRAIN'     /* schema name          */
,'STATS'     /* statistics table name */
,'USERS'     /* tablespace           */
);
```

Example: Copying Statistics

Use the `CREATE_STAT_TABLE` procedure in the package `DBMS_STATS` to create the user-defined `STATS` table that will hold the statistics.

The statistics in this table are not accessible by the Oracle optimizer, and thus cannot be used for generating an explain plan. Likewise, future commands to analyze the data, will not update the information held in this table.

Example: Copying Statistics

Step 2. Copy the statistics into the table:

```
DBMS_STATS.EXPORT_TABLE_STATS
('TRAIN'      /* schema name          */
,'COURSES'    /* table name           */
, NULL        /* no partitions        */
,'STATS'      /* statistics table name */
,'CRS990601'  /* id for statistics    */
, TRUE        /* index statistics     */
);
```

Step 3. Export the STATS table, and then import it into the second database.

Copying Statistics from the Data Dictionary to a User-Defined Table

To copy statistics for a table from the data dictionary to a user-defined statistics table, use one of the following procedures:

- EXPORT_COLUMN_STATS
- EXPORT_INDEX_STATS
- EXPORT_SYSTEM_STATS
- EXPORT_TABLE_STATS
- EXPORT_SCHEMA_STATS
- EXPORT_DATABASE_STATS

The example on the slide is showing an export of the statistics for the COURSES table only.

Example: Copying Statistics

Step 4. Copy the statistics into the data dictionary:

```
DBMS_STATS.IMPORT_TABLE_STATS
('TRAIN'      /* schema name          */
,'COURSES'    /* table name           */
, NULL        /* no partitions        */
,'STATS'      /* statistics table name */
,'CRS990601'  /* id for statistics     */
, TRUE        /* index statistics      */
);
```

Copying Statistics from a User Defined Table to the Data Dictionary

To copy statistics for a table from a user-defined statistics table into the data dictionary, use one of the following procedures:

- `IMPORT_COLUMN_STATS`
- `IMPORT_INDEX_STATS`
- `IMPORT_SYSTEM_STATS`
- `IMPORT_TABLE_STATS`
- `IMPORT_SCHEMA_STATS`
- `IMPORT_DATABASE_STATS`

The example on the slide is showing an import of the statistics for the `COURSES` table only.

Summary

In this lesson, you should have learned how to:

- Describe how the optimizer is used
- Explain the concept of plan stability
- Explain the use of stored outlines
- Describe how hints are used
- Use SQL Trace and `TKPROF`
- Collect statistics on indexes and tables
- Describe the use of histograms
- Copy statistics between databases

Practice 14

The objective of this practice is to familiarize you with SQL statement execution plans and to interpret the formatted output of a trace file generated using SQL Trace and the formatted output generated by TKPROF.

1. Connect as hr/hr, and create the PLAN_TABLE under the HR schema, if it is not already created, by running $ORACLE_HOME/rdbms/admin/utlxplan.sql

   Note: If plan_table already exists and holds rows truncate the table.

2. Set the Optimizer_goal to rule based using the alter session command, and generate the explain plan for the statement $HOME/STUDENT/LABS/lab14_02.sql. View the generated plan by querying object_name, operation, optimizer from PLAN_TABLE.

3. Truncate the PLAN_TABLE. Change the optimizer_goal to cost based by setting the value to ALL_ROWS, and rerun the explain plan for $HOME/STUDENT/LABS/lab14_02.sql. Notice that the Optimizer mode, and the explain plan have changed.

   Note: Although exactly the same scripts are being run, due to the different optimizer settings, different explain paths are found. With rule based, one of the rules is to use any index that is on the columns in the where clause. By using cost based optimizer mode, the server has been able to determine that it will be faster to just perform a full table scan, due to the number of rows being returned by the script.

4. Truncate the PLAN_TABLE, and set the optimizer goal to rule by using the alter session command. This time generate the explain plan for the script $HOME/STUDENT/LABS/lab14_04.sql. Examine the script which is a copy of $HOME/STUDENT/LABS/lab14_02.sql except it changes the line "select *" to include a hint /*+ all_rows*/ for the optimizer. View the generated execution plan by querying object_name, operation, optimizer from PLAN_TABLE.

5. Exit out of SQLPLUS, change the directory to $HOME/ADMIN/UDUMP and delete all the trace files already generated.

6. Connect as sh/sh and enable SQL TRACE, using the alter session command, to collect statistics for the script, $HOME/STUDENT/LABS/lab14_05.sql. Run the script. After the script has completed, disable the SQL_TRACE. and then format your trace file using TKPROF. Use the options SYS=NO and EXPLAIN= sh/sh. Name the file myfile.txt

7. View the output file myfile.txt, and note the CPU, current, and query figures for the fetch phase. Do not spend time analyzing the contents of this file as the only objective here is to become familiar and comfortable with running TKPROF and SQL Trace.

8. Connect hr/hr and gather statistics for all objects under the HR schema using the DBMS_STATS package, while saving the current statistics then restore the original statistics.
   a. Connect as HR and create a table to hold statistics in that schema.
   b. Save the current schema statistics into your local statistics table.
   c. Analyze all objects under the HR schema.
   d. Remove all schema statistics from the dictionary and restore the original statistics you saved in step b.

Quick Reference

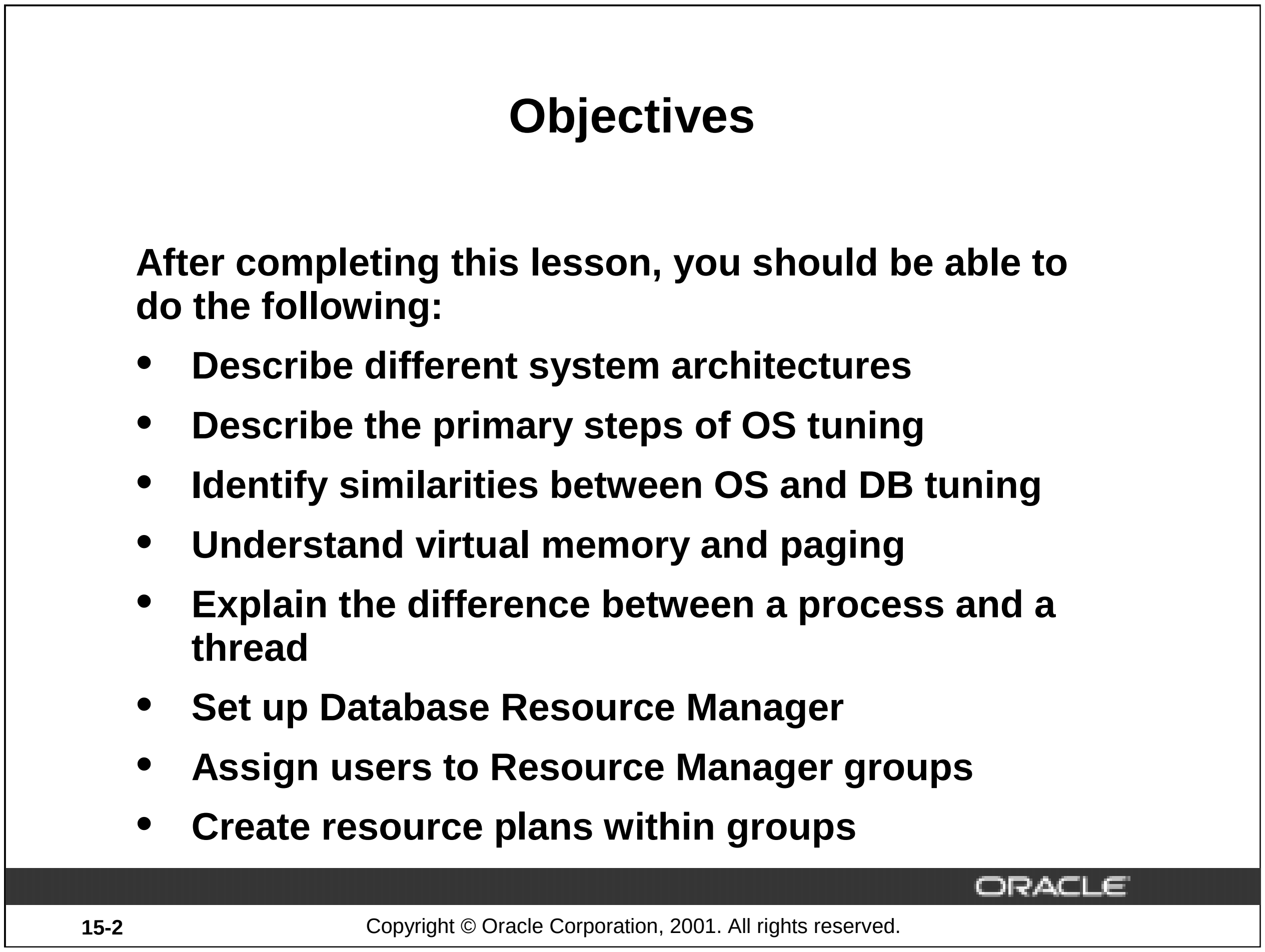
Context	Reference
Parameters	CREATE_STORED_OUTLINES MAX_DUMP_FILE_SIZE USER_DUMP_DEST TIMED_STATISTICS SQL_TRACE USE_STORED_OUTLINES OPTIMIZER_MODE QUERY_REWRITE_ENABLED QUERY_REWRITE_INTEGRITY
Dynamic performance views	
Data dictionary views	Various USER_*, ALL_*, and DBA_* views

Commands	ALTER SYSTEM
	ALTER SESSION
	ANALYZE
	CREATE BITMAP INDEX
	CREATE INDEX ... REVERSE
	CREATE MATERIALIZED VIEW
	CREATE OUTLINE
	CREATE TABLE ... ORGANIZATION INDEX
	EXPLAIN PLAN

Procedures	SYS.DBMS_MVIEW package SYS.DBMS_OLAP package SYS.DBMS_SESSION.SET_SQL_TRACE SYS.DBMS_SYSTEM.SET_SQL_TRACE_IN_SESSION SYS.DBMS_STATS package SYS.OUTLN_PKG package

15

Tuning the Operating System and Using Resource Manager



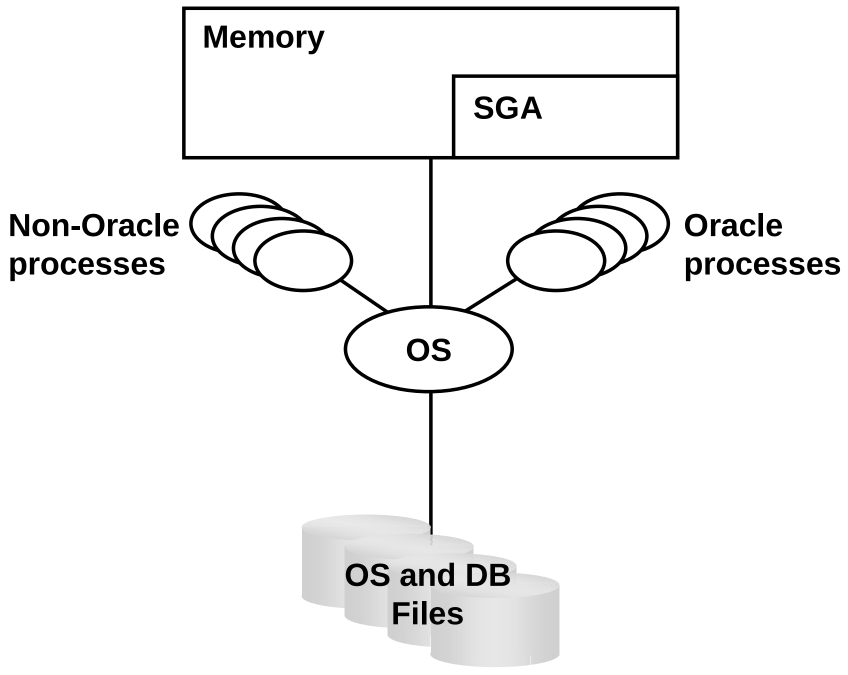
Objectives

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

- Describe different system architectures
- Describe the primary steps of OS tuning
- Identify similarities between OS and DB tuning
- Understand virtual memory and paging
- Explain the difference between a process and a thread
- Set up Database Resource Manager
- Assign users to Resource Manager groups
- Create resource plans within groups

15-2

Operating System Tuning



Introduction to Operating System Tuning

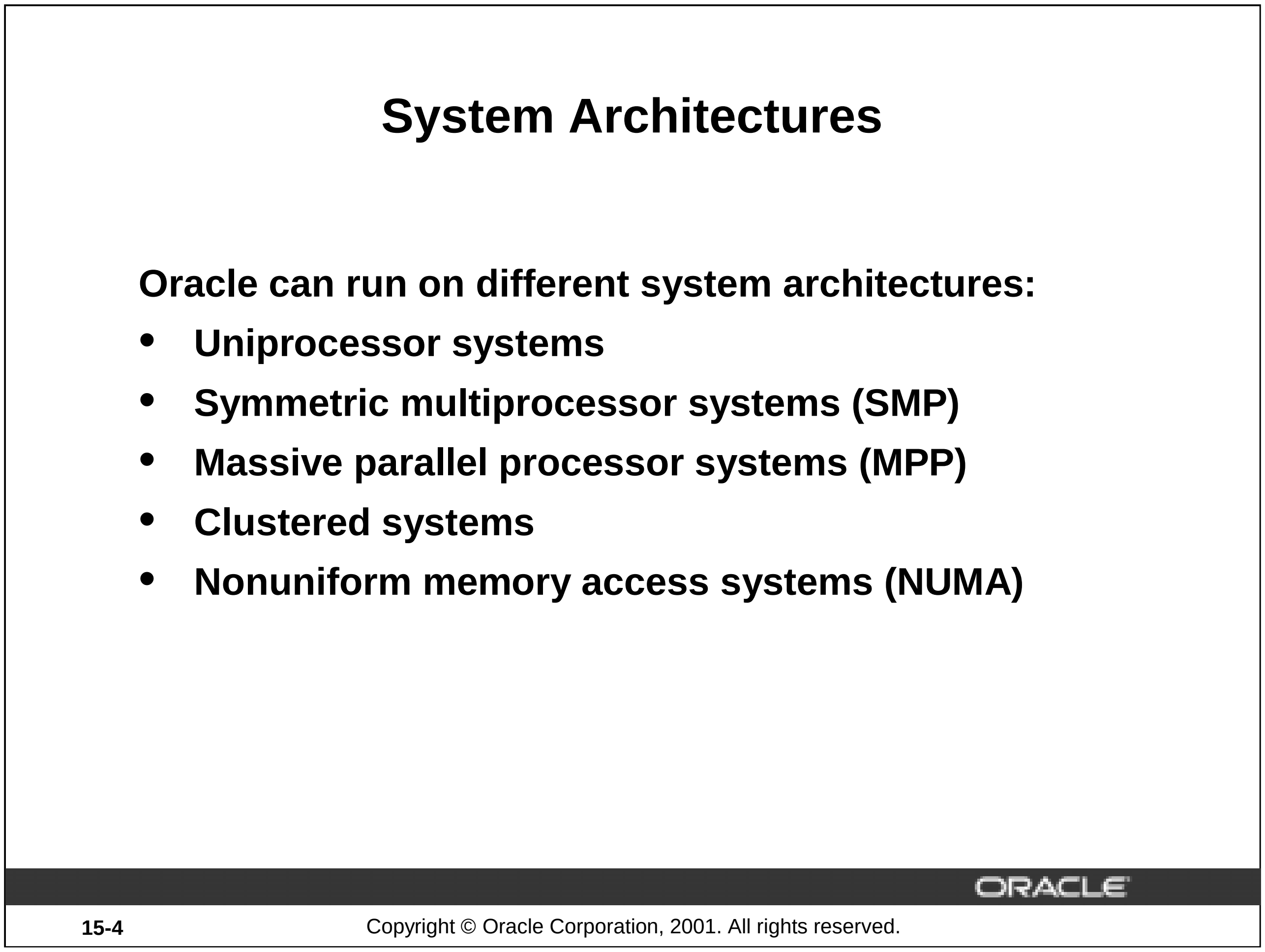
The system administrator, the person responsible for tuning the operating system (OS), has tuning concerns similar to those of the database administrator. But the system administrator is also concerned with how applications other than Oracle applications affect performance. When tuning, the system administrator considers:

- Memory usage
- I/O levels
- CPU usage
- Network traffic

This lesson gives an overview of OS tuning, not specifics. Network tuning is not included, because it is a distributed system consideration. This class focuses on OS tuning as it relates to the Oracle database rather than system performance tuning issues.

The operating system is tuned in a specific order because each area has its effect on the other underlying areas. If the memory usage is too high for example, an extra load is placed on the I/O layer, which in turn places an extra load on the CPU. The correct tuning order is:

1. Memory
2. I/O
3. CPU



System Architectures

Oracle can run on different system architectures:

- **Uniprocessor systems**
- **Symmetric multiprocessor systems (SMP)**
- **Massive parallel processor systems (MPP)**
- **Clustered systems**
- **Nonuniform memory access systems (NUMA)**

Uni Processor systems

Uni Processor systems have only one CPU and one memory.

Symmetric Multi Processor (SMP) systems

SMP systems have multiple CPUs. The number commonly ranges from two to 64. All of the CPUs in an SMP machine share the same memory, system bus, and I/O system. A single copy of the operating system controls all of the CPUs.

Massively Parallel Processor (MPP) systems

MPP systems consist of several nodes connected together. Each node has its own CPU, memory, bus, disks, and I/O system. Each node runs its own copy of the operating system. The number of nodes in an MPP system can range from two to even several thousands.

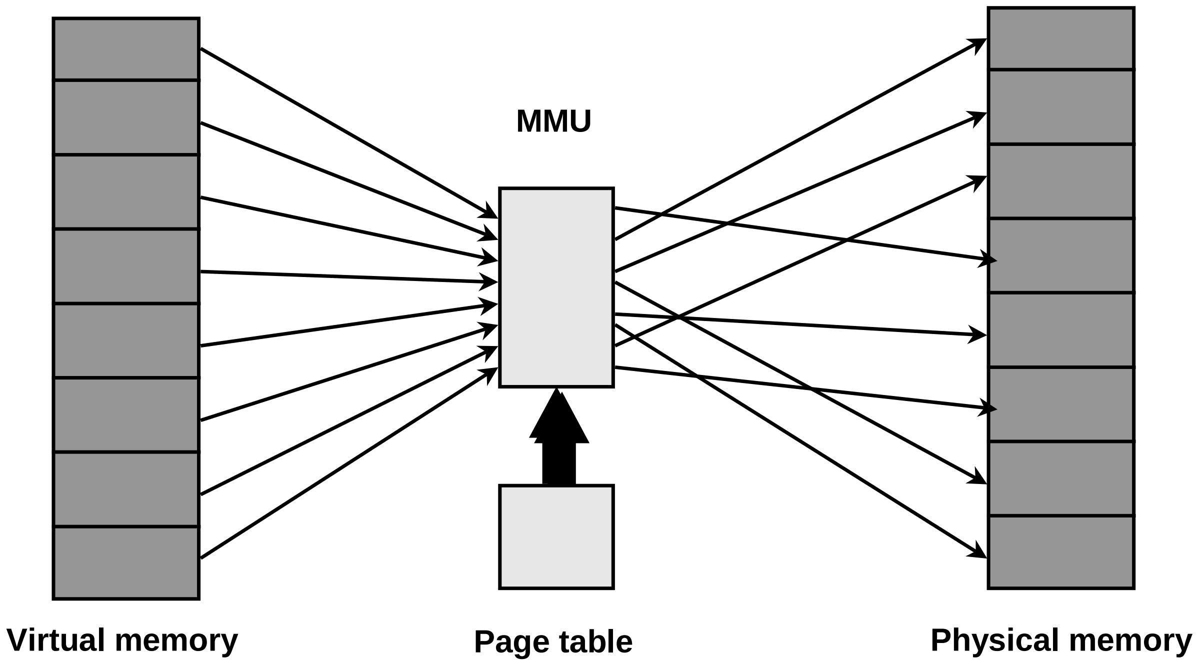
Non Uniform Memory Access (NUMA) systems

NUMA systems consist of several SMP systems that are interconnected in order to form a larger system. In contrast to a cluster all of the memory in all of the SMP systems are connected together to form a single large memory space transparent to all sub-systems. A single copy of the operating system runs across all nodes.

Clustered (Cluster) systems

A cluster consist of several nodes *loosely* coupled using local area network (LAN) interconnection technology. Each of the individual nodes can in itself be composed of a single-processor machine or SMP machine. In a cluster, system software balances the workload among the nodes and provides for high availability.

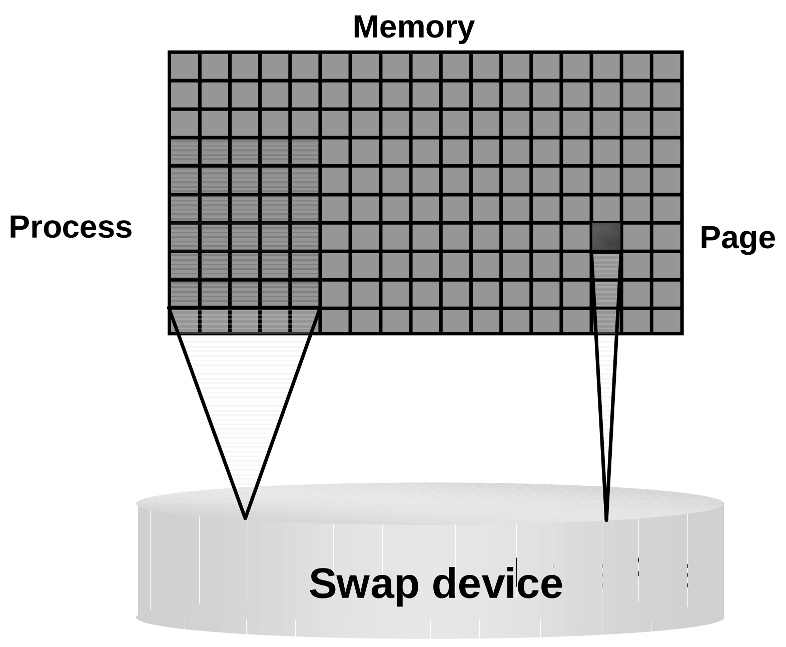
Virtual and Physical Memory



Virtual Memory

Operating systems make use of *virtual memory*. Virtual memory gives the application the feeling that they are the only application on the system. Each application *sees* a complete isolate memory area starting at address zero. This virtual memory area is divided into memory pages which are usually 4 or 8 KB in size. The Operating System *maps* these virtual memory pages into physical memory by the use of a memory management unit (MMU). The mapping between virtual and physical memory is under control of a *page table*. On most operating systems, each process has its own page table. This can cause memory wastage if many processes need to access a very large area of *shared memory*. (read: Oracle) On some platforms, Solaris for example, this memory wastage can be avoided by sharing the page table entries for a shared memory area. This is called intimate shared memory (ISM). An additional benefit of using ISM is that the shared memory area gets locked into physical memory.

Paging and Swapping



Paging and Swapping

Operating Systems use the same technique to manage memory as Oracle: they try to keep the most recently used pages in real memory. Inadequate memory resources cause too much paging or swapping to occur. This symptom is often called thrashing, because it causes blocks to be transferred back and forth (thrashed) between memory and disk.

Paging occurs when a process needs a page (block) of memory that is no longer in real memory but in virtual memory. The block must be read (paged) in from the disk, and the block in memory that it replaces may also need to be written to disk. Swapping is similar to paging, except that the memory space of the entire process is removed from memory. If there are too many processes running at a time, swapping may increase to an unacceptable level.

Both swapping and paging require adequate disk space to temporarily hold the memory blocks on disk. These files are I/O intensive, so they also need to be considered when balancing I/O. Some operating systems such as Microsoft Windows NT, 2000 do not use swapping only paging, and most others are swapping only as a last resort when the amount of free memory is really getting unacceptably low.

Tuning Memory

- **Database tuning can improve paging performance by locking SGA into real memory.**
- **The DBA should monitor real and virtual memory use.**
- **The DBA should use intimate shared memory, if it is available.**

DB Tuning and Its Effects on Paging

Besides tuning the SGA, the DBA can also affect paging and swapping performance in another way.

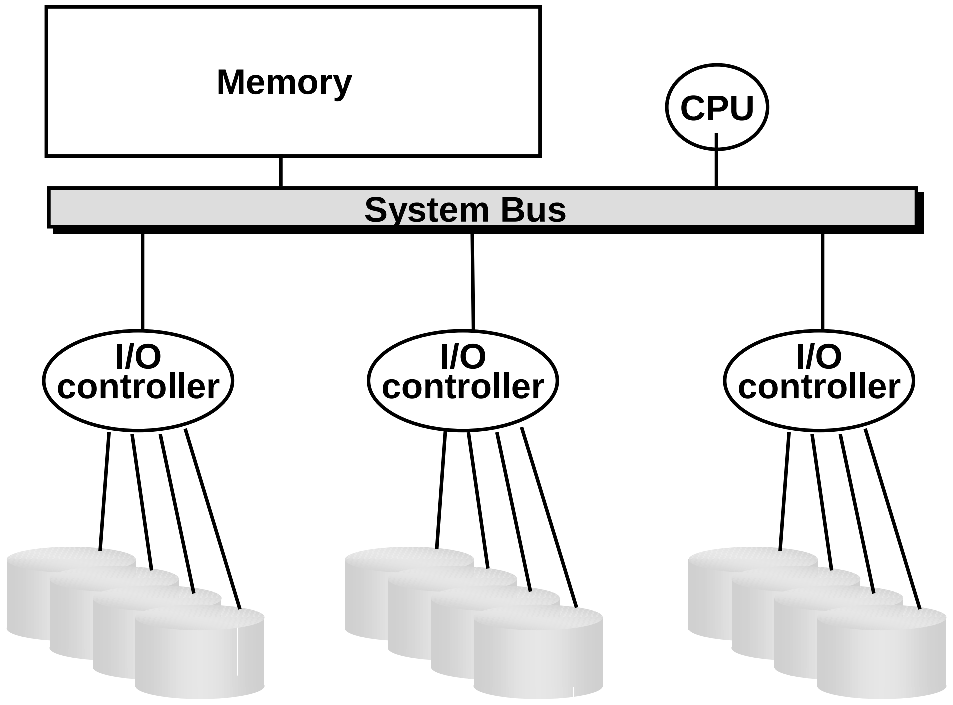
On some operating systems, the DBA can lock the SGA into real memory by setting the LOCK_SGA initialization parameter to TRUE, so it is never paged out to disk. Obviously, the Oracle server performs better if the entire SGA is kept in real memory.

This should be used only on systems that have sufficient memory to hold all the SGA pages without degrading performance in other areas.

Monitor Memory Usage

Real and virtual memory usage and paging and swapping can usually be monitored by process or for the entire operating system. The amount of paging and swapping that is acceptable varies by operating system; some tolerate more than others.

Tuning I/O



Tuning I/O

The system administrator improves the performance of disk I/O by balancing the load across disks and disk controllers.

I/O-intensive systems, such as database servers, perform better with many small disks instead of a few large disks. More disks reduce the likelihood that a disk becomes a bottleneck. Parallel Query operations also benefit by distributing the I/O workload over multiple disk drives.

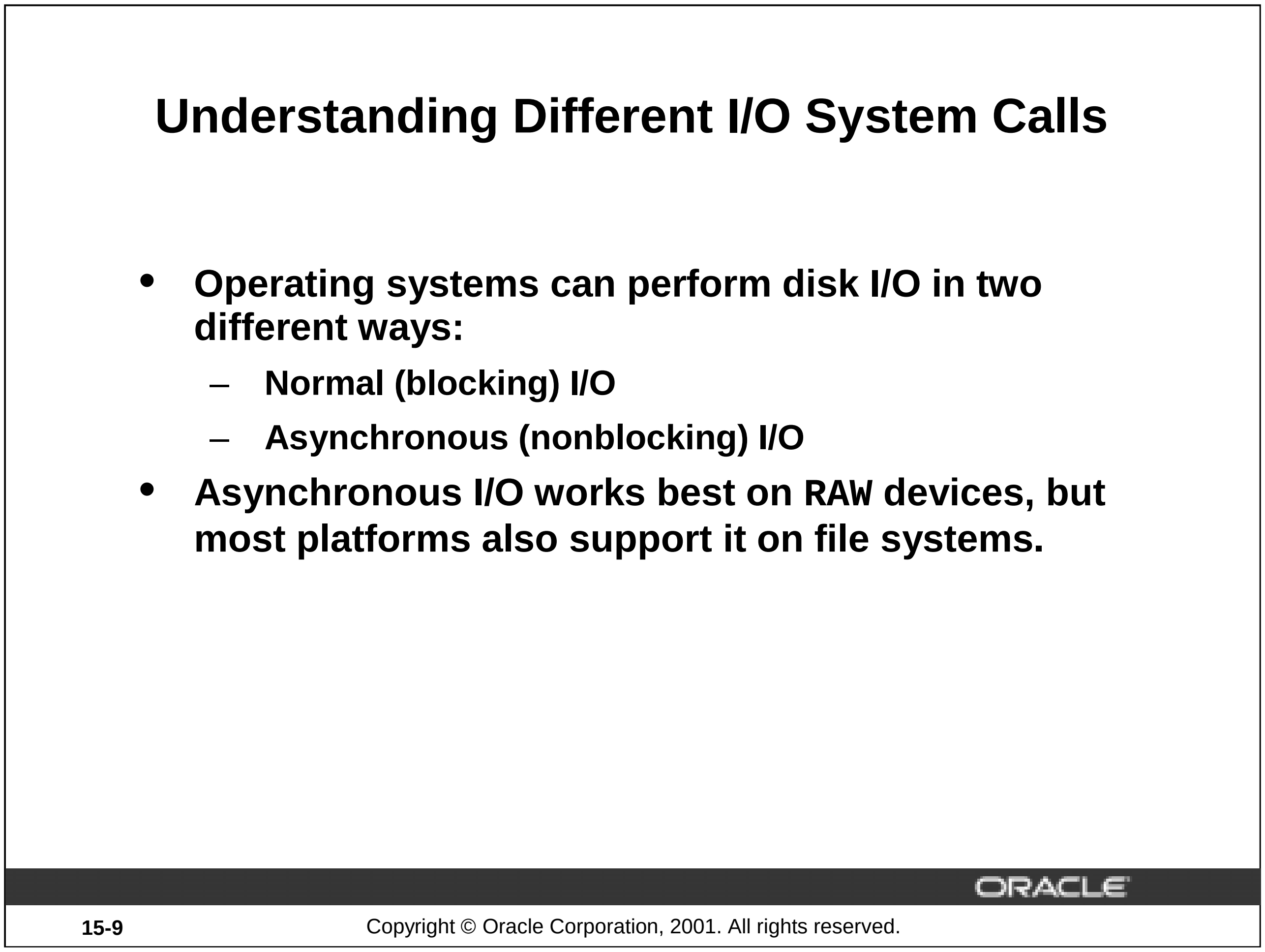
Raw Devices

A raw device is a disk or disk partition without a file or directory structure. Although they are more difficult to administer than operating system files, they can provide some performance benefit, because reads and writes to a raw device bypass the operating system cache.

On some operating systems, such as Windows NT, Oracle server process does not use the O/S file system cache in I/O operations. In these cases, use of raw devices may not show significant performance gains.

Monitoring

I/O performance statistics usually include the number of reads and writes, reads and writes per second, and I/O request queue lengths. Acceptable loads vary by device and controller.



Understanding Different I/O System Calls

- **Operating systems can perform disk I/O in two different ways:**
  - **Normal (blocking) I/O**
  - **Asynchronous (nonblocking) I/O**
- **Asynchronous I/O works best on `RAW` devices, but most platforms also support it on file systems.**

15-9

The Normal (or blocking) I/O system call

When Oracle issues an I/O request (read or write) using a normal (or blocking) I/O system call, it has to wait until the I/O operation has completed. This limits the amount of I/O Oracle can perform in a certain amount of time.

The Asynchronous (or non-blocking) I/O system call

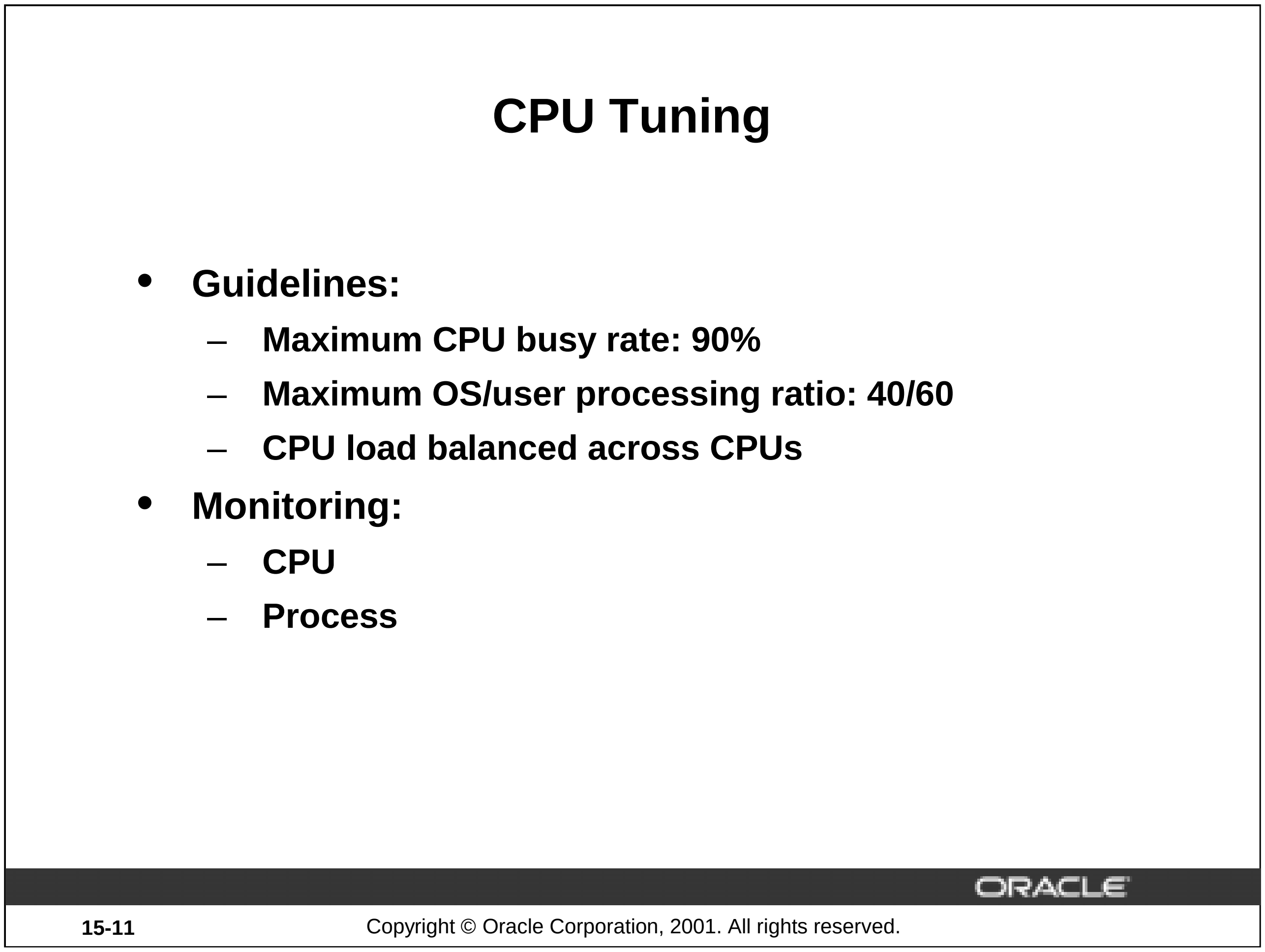
When Oracle issues an I/O request (read or write) using a asynchronous (or non-blocking) I/O system call, it can continue processing and doesn't have to wait until the I/O operation completes. This allows Oracle to issue many I/O requests at the same time. By using asynchronous I/O Oracle can overlap several I/O requests. Once the operating system has completed an asynchronous I/O system call it will notify Oracle about the fact that the I/O request has been completed along with with the status of this particular I/O request.

Asynchronous I/O on file systems

Although asynchronous I/O on file systems is supported on most UNIX implementations, it is usually implemented using the user-level multithreading capabilities in the operating system. Therefore, it induces a significant CPU overhead. In order to avoid this CPU overhead it is preferred to use multiple database writers or use database writer I/O slaves rather than asynchronous I/O.

Asynchronous I/O on RAW devices

In contrast to asynchronous I/O on file systems, asynchronous I/O on RAW-devices is implemented in the OS by using kernel threads, without significant CPU overhead. An additional benefit is that the data is not buffered in the operating system cache which reduces memory requirements.



CPU Tuning

- **Guidelines:**
  - **Maximum CPU busy rate: 90%**
  - **Maximum OS/user processing ratio: 40/60**
  - **CPU load balanced across CPUs**
- **Monitoring:**
  - **CPU**
  - **Process**

CPU Tuning Guidelines

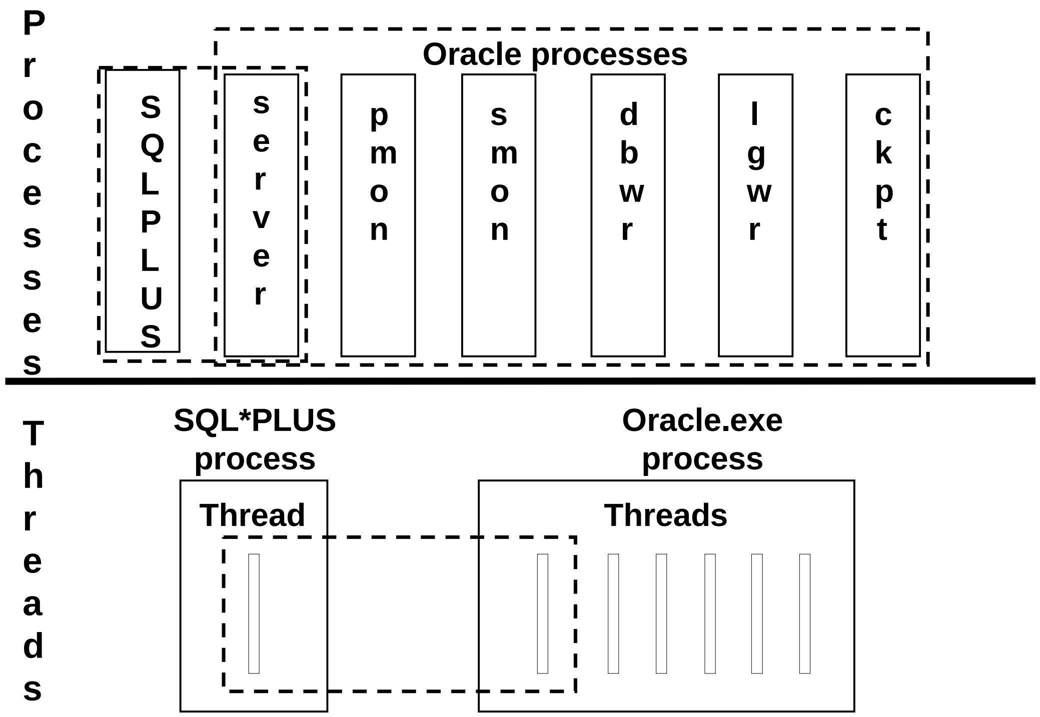
When tuning CPU usage, the system administrator has these primary concerns:

- Are there adequate CPU resources? The system administrator ensures that the CPU is not too busy. As a general rule, if the CPU is busy 90% of the time, it has probably reached its capacity.
- Is there a good ratio between operating system processing and application processing? Operating system processing includes the tasks that the operating system performs for the applications; for example, reading and writing to devices, sending messages between processes, and scheduling processes.
- The goal is to have have the CPU working mostly on the application, and least on operating system related tasks. Too much time spent in the operating system mode is a symptom of an underlying problem, such as:
  - Insufficient memory, which also causes swapping or paging
  - Poor application design, causing too many operating system calls
- For multiprocessor systems, the system administrator must also check that the CPU load is balanced across all of the CPUs, particularly if any of the CPUs have a very high usage rate.

CPU Monitoring

Operating system monitors for CPU usage normally include the percentage of time the CPU is active and the time spent in operating system versus user tasks. Process monitors show the process status and statistics on the number of I/Os, operating system calls, and paging and swapping rates.

Process versus Thread



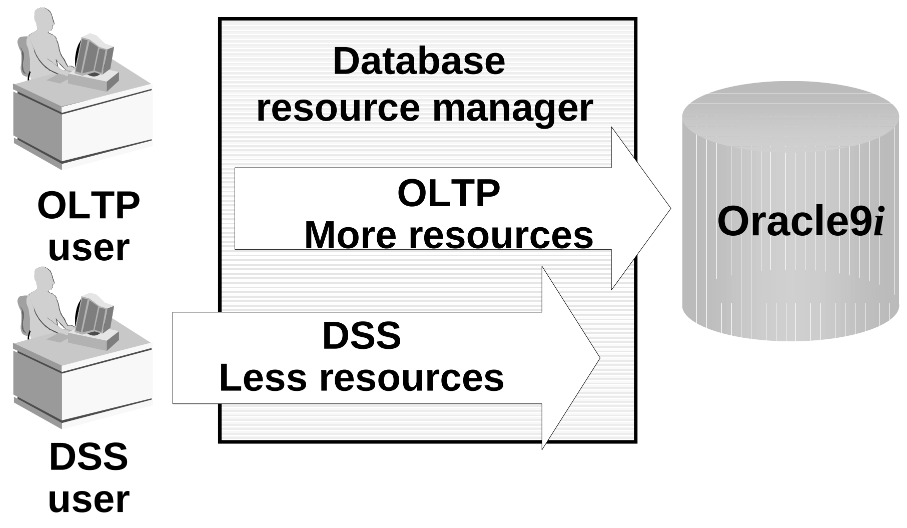
Processes and Threads

On most operating systems, each Oracle process is also an operating system process. However under some operating systems, notably Microsoft Windows NT and 2000, Oracle is implemented as a single operating system process with multiple threads. In Windows NT, Oracle processes threads share the memory allocated to the process.

Each Oracle process is a thread within the operating system process. A thread is a sequence of instructions that can execute independently of other threads in the same process. This configuration makes SGA access and communication among Oracle processes more efficient at the expense of having a limit on the maximum process memory.

Overview of Database Resource Manager

- Manage mixed workload
- Control system performance



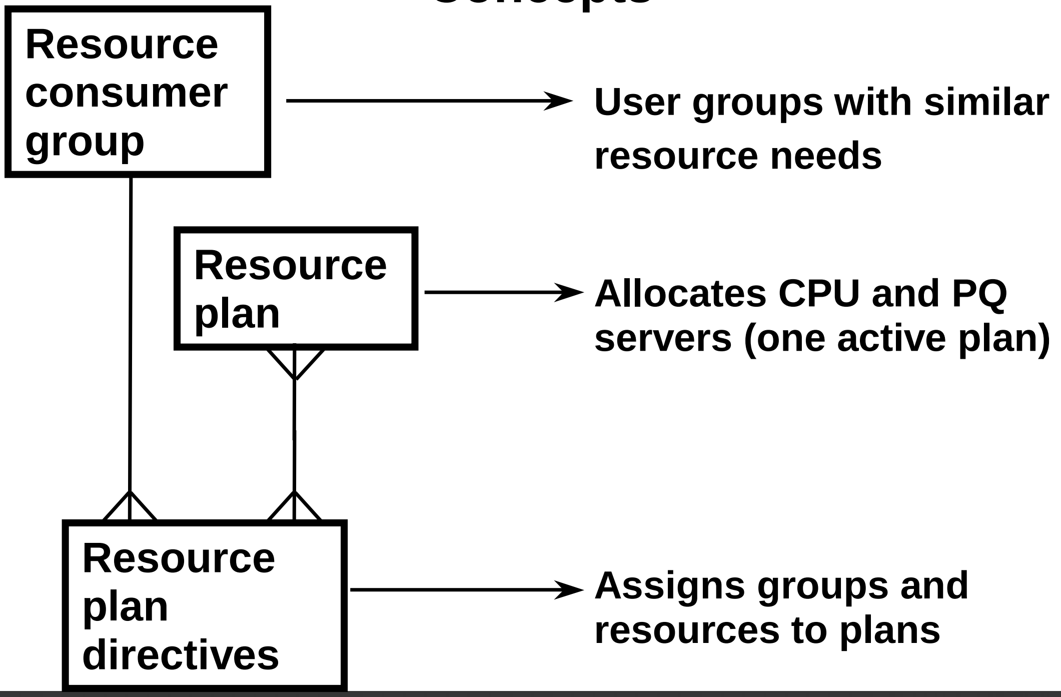
Overview

The Database Resource Manager allows the DBA to have more control over certain resources utilization than is normally possible through operating system resource management alone. With Oracle9i it is possible to have control over CPU utilization and degree of parallelism. Improved resource management enables better application performance and availability. The RDBMS has traditionally left resource management decisions in the hands of the operating system causing inefficient scheduling, mostly descheduling, of Oracle servers while they hold latches.

By using the Database Resource Manager, the database administrator can:

- Guarantee groups of users a minimum amount of processing resources regardless of the load on the system and the number of users
- Distribute available processing resources, by allocating percentages of CPU time to different users and applications. In an OLTP environment, a higher priority can be given to OLTP applications than to DSS applications during normal business hours.
- Limit the degree of parallelism that a set of users can use
- Configure an instance to use a particular plan for allocating resources. A DBA can dynamically change the method, for example, from a daytime setup to a nighttime setup, without having to shutdown and restart the instance.

Database Resource Management Concepts


Resource consumer group
User groups with similar resource needs
Resource plan
Allocates CPU and PQ servers (one active plan)
Resource plan directives
Assigns groups and resources to plans

Database Resource Manager Concepts

Administering systems using the database resource management involves the use if resource plans, resource consumer groups, and resource plan directives.

Resource Consumer Group

Resource consumer groups define a set of users who have similar requirements for resource use, and also specifies a resource allocation method for allocating the CPU among sessions. To control resource consumption, you can assign user sessions to resource consumer groups. A user can be assigned to multiple consumer groups but only one group can be active at a time for a session, and either the user or DBA can switch the consumer group during a session.

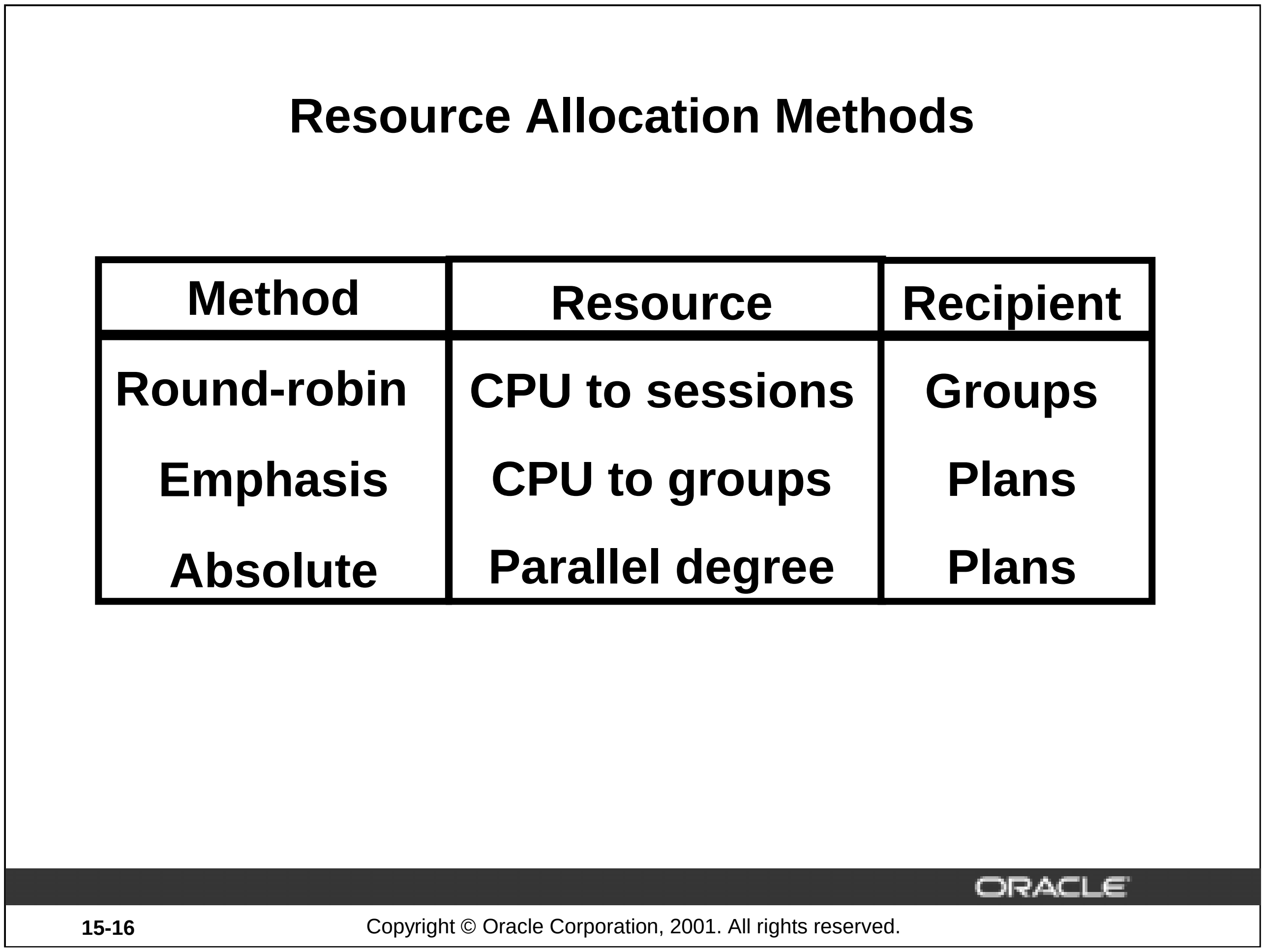
Resource Plan

Resource allocations are specified in a resource plan. Resource plans contain resource plan directives, which specify the resources that are to be allocated to each resource consumer group.

Resource Plan Directives

There is one resource plan directive for each entry in the plan. These directives are a means of:

- Assigning consumer groups or subplans to resource plans
- Allocating resources among consumer groups in the plan by specifying parameters for each resource allocation method



Resource Allocation Methods

Method	Resource	Recipient
Round-robin	CPU to sessions	Groups
Emphasis	CPU to groups	Plans
Absolute	Parallel degree	Plans

15-16

Resource Allocation Methods

Resource allocation methods determine the method that Database Resource Manager uses when allocating a particular resource to a resource consumer group or resource plan. Currently, the resource allocation method for allocating the CPU among resource consumer groups is the emphasis method. The only resource allocation method for limiting the degree of parallelism is the absolute method.

CPU Allocation between Sessions in a Group: Round-Robin Method

Inside each consumer group, the CPU resources are distributed in a round-robin fashion.

Parallel Degree Limit for Resource Plans: Absolute Method

This limits the maximum degree of parallelism of any operation. This parameter is only allowed in directives that refer to resource consumer groups. Currently, the absolute method is the only one possible. It specifies how many processes may be assigned to an operation. If there are multiple plan directives referring to the same subplan or consumer group, the parallel degree limit for that subplan or consumer group will be the minimum of all the incoming values.

CPU Allocation between Groups in a Plan: Emphasis Method

The emphasis CPU allocation method determines how much emphasis is given to sessions in different consumer groups in a resource plan. CPU usage is assigned levels from 1 to 8, with level 1 having the highest priority. Percentages specify how to allocate CPU to each consumer group at each level. The following rules apply for the emphasis resource allocation method:

- Sessions in resource consumer groups with non-zero percentages at higher-priority levels always get the first opportunity to run.
- CPU resources are distributed at a given level based on the specified percentages. The percentage of CPU specified for a resource consumer group is a maximum for how much that consumer group can use at a given level. If any CPU resources are left after all resource consumer groups at a given level have been given an opportunity to run, the remaining CPU resources fall into to the next level.
- The sum of percentages at any given level must be less than or equal to 100.
- Any unused CPU time gets recycled. In other words, if no consumer groups are immediately interested in a specific period of CPU time (due to percentages), the consumer groups get another opportunity to use the CPU time, starting at level one.
- Any levels that have no plan directives explicitly specified have a default of 0% for all subplans or consumer groups.
- The emphasis resource allocation method avoids starvation problems.

The previous rules apply under the following assumptions: The Database Resource Manager percentage limits are not hard limits. It is certain to act as limits only if system throughput is at maximum. If less than maximum, resource consumer groups can be provided the resources they demand, even if that means more than the specified limit, provided the consumer groups in question do not have other higher priority groups requesting the spare resources. Database Resource Manager first seeks to maximize throughout, then to prioritize among the consumer groups.

Distribution of Resources to Consumer Groups vs. Limitations on Private Usage

Profiles define resource limits for individual users or sessions. An example is the idle_time resource limit. A resource consumer group is a set of users treated as a collective unit by the resource manager. The resource plan directives assign resources to consumer groups as a whole, not to individual sessions, whereas profiles limit the resources that individual sessions can consume regardless of other sessions that might be executing.

The Original Plan: SYSTEM_PLAN

Resource consumer group	Allocation methods			
	P1CPU	P2CPU	P3CPU	P1 //
SYS_GROUP	100%	0%	0%	0
OTHER_GROUPS	0%	100%	0%	0
LOW_GROUP	0%	0%	100%	0

The Original Plan: `SYSTEM_PLAN`

Oracle provides the *SYSTEM_PLAN* as the original resource plan. This plan contains directives for the following provided consumer groups :

`SYS_GROUP`

Is the initial consumer group for the users `SYS` and `SYSTEM`

`OTHER_GROUPS`

Is used for all sessions who belong to consumer groups that are not part of the active resource plan. There must be a plan directive for `OTHER_GROUPS` somewhere in any active plan.

`LOW_GROUP`

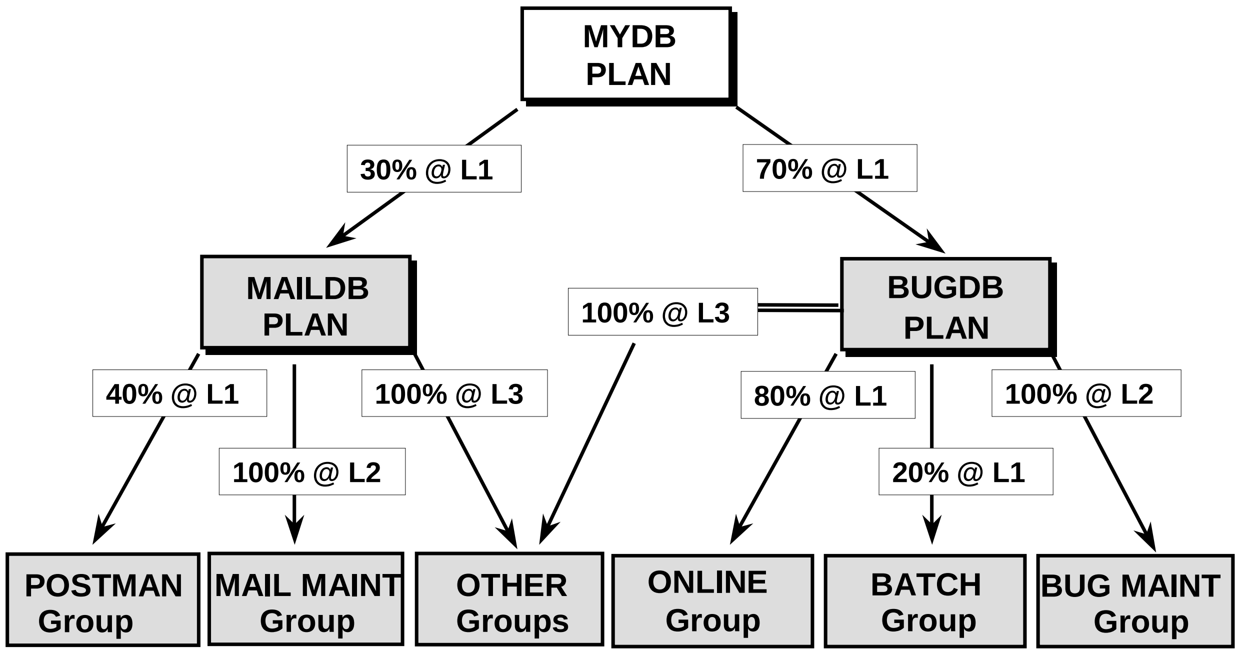
Provides a group having lower priority than `SYS_GROUP` and `OTHER_GROUPS` in this plan. It is up to you to decide which user sessions will be part of `LOW_GROUP`. Initially, no user is associated to this consumer group. Note that *switch* privilege is granted to `PUBLIC` for this group.

`Initial Resource Consumer Group`

The initial consumer group of a user is the consumer group to which any session created by that user initially belongs. If you have not set the initial consumer group for a user, the user's initial consumer group will automatically be the `DEFAULT_CONSUMER_GROUP`.

Note: This setting will simulate a priority system.

Using Subplans



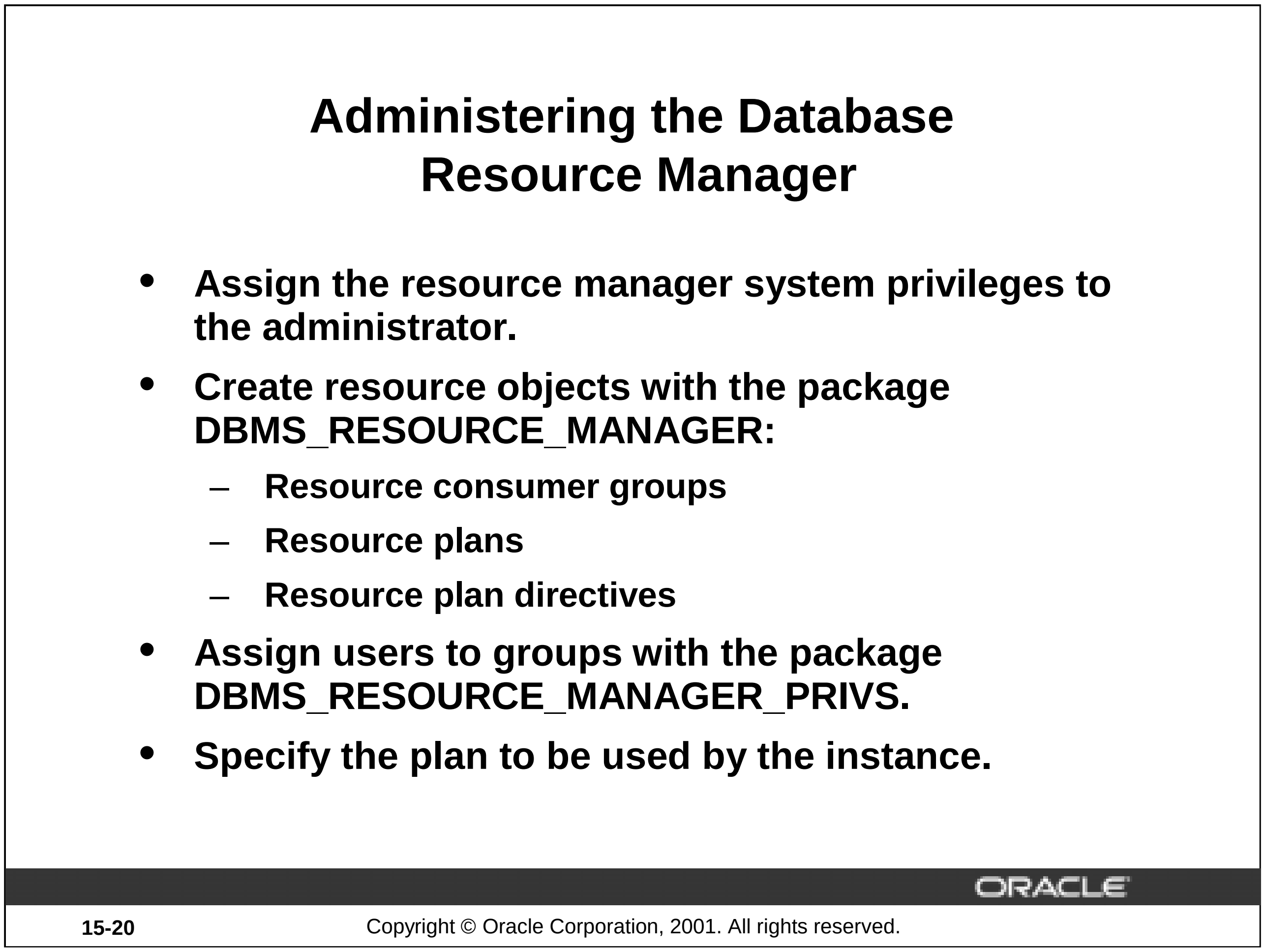
Subplan

A plan cannot only contain resource consumer groups, it can also contain other plans, called *subplans*. It is possible for a subplan or consumer group to have more than one parent (owning plan), but there cannot be any loops in a plan.

Example

If the MYDB resource plan were in effect and there were an infinite number of runnable users in all resource consumer groups, the MAILDB plan would be in effect 30% of the time, while the BUGDB plan would be in effect 70% of the time.

Moreover, if the MAILDB plan allocates 40% of resources to the Postman consumer group and the BUGDB plan allocates 80% of resources to the Online consumer group, then users in the Postman group would be run 12% (40% of 30%) of the time, while users in the Online group would be run 56% (80% of 70%) of the time.



Administering the Database Resource Manager

- **Assign the resource manager system privileges to the administrator.**
- **Create resource objects with the package DBMS_RESOURCE_MANAGER:**
  - **Resource consumer groups**
  - **Resource plans**
  - **Resource plan directives**
- **Assign users to groups with the package DBMS_RESOURCE_MANAGER_PRIVS.**
- **Specify the plan to be used by the instance.**

ORACLE

15-20

Administering the Database Resource Manager

In order to administer the Database Resource Manager, you must have the `ADMINISTER_RESOURCE_MANAGER` system privilege. DBAs have this privilege with the `ADMIN` option because it is part of the DBA role. Being an administrator for the Database Resource Manager allows you to execute all of the procedures in the `DBMS_RESOURCE_MANAGER` package.

The `ADMINISTER_RESOURCE_MANAGER` privilege is granted and revoked with the `DBMS_RESOURCE_MANAGER_PRIVS` package. It cannot be granted through the SQL `grant` or `revoke` statements.

`DBMS_RESOURCE_MANAGER` Procedures

- `CREATE_PLAN`: Names a resource plan and specifies its allocation methods.
- `UPDATE_PLAN`: Updates a resource plan's comment.
- DELETE_PLAN : Deletes a resource plan and its directives.
- `DELETE_PLAN_CASCADE`: Deletes a resource plan and all of its descendents.
- `CREATE_CONSUMER_GROUP`: Names a resource consumer group and specifies its allocation method.
- `UPDATE_CONSUMER_GROUP`: Updates a consumer group's comment.
- `DELETE_CONSUMER_GROUP`: Deletes a consumer group.

DBMS_RESOURCE_MANAGER Procedures (continued)

- CREATE_PLAN_DIRECTIVE: Specifies the resource plan directives that allocate resources to resource consumer groups within a plan or among subplans in a multilevel *plan schem*a.
- UPDATE_PLAN_DIRECTIVE: Updates plan directives.
- DELETE_PLAN_DIRECTIVE: Deletes plan directives.
- CREATE_PENDING_AREA: Creates a pending area (scratch area) within which changes can be made to a plan schema.
- VALIDATE_PENDING_AREA: Validates the pending changes to a plan schema.
- CLEAR_PENDING_AREA: Clears all pending changes from the pending area.
- SUBMIT_PENDING_AREA: Submits all changes to a plan schema.
- SET_INITIAL_CONSUMER_GROUP: Sets the initial consumer group for a user.
- SWITCH_CONSUMER_GROUP_FOR_SESS: Switches the consumer group of a specific session.
- SWITCH_CONSUMER_GROUP_FOR_USER: Switches the consumer group of all sessions belonging to a specific user.

DBMS_RESOURCE_MANAGER_PRIVS Procedures

- GRANT_SYSTEM_PRIVILEGE: Grants ADMINISTER_RESOURCE_MANAGER system privilege to a user or role.
- REVOKE_SYSTEM_PRIVILEGE: Revokes ADMINISTER_RESOURCE_MANAGER system privilege to a user or role.
- GRANT_SWITCH_CONSUMER_GROUP: Grants permission to a user, role, or PUBLIC to switch to a specified resource consumer group.
- REVOKE_SWITCH_CONSUMER_GROUP: Revokes permission for a user, role, or PUBLIC to switch to a specified resource consumer group.

The above description gives you an overview of the possibilities. In order to have a comprehensive description of the various procedures you should refer to the Oracle*9i* Supplied PL/SQL Packages Reference.

In the rest of this Lesson, we will be looking at how to set up the Database Resource Manager by using the PL/SQL packages.

Assigning the Resource Manager Privilege

Assign the resource manager system privileges to the administrator.

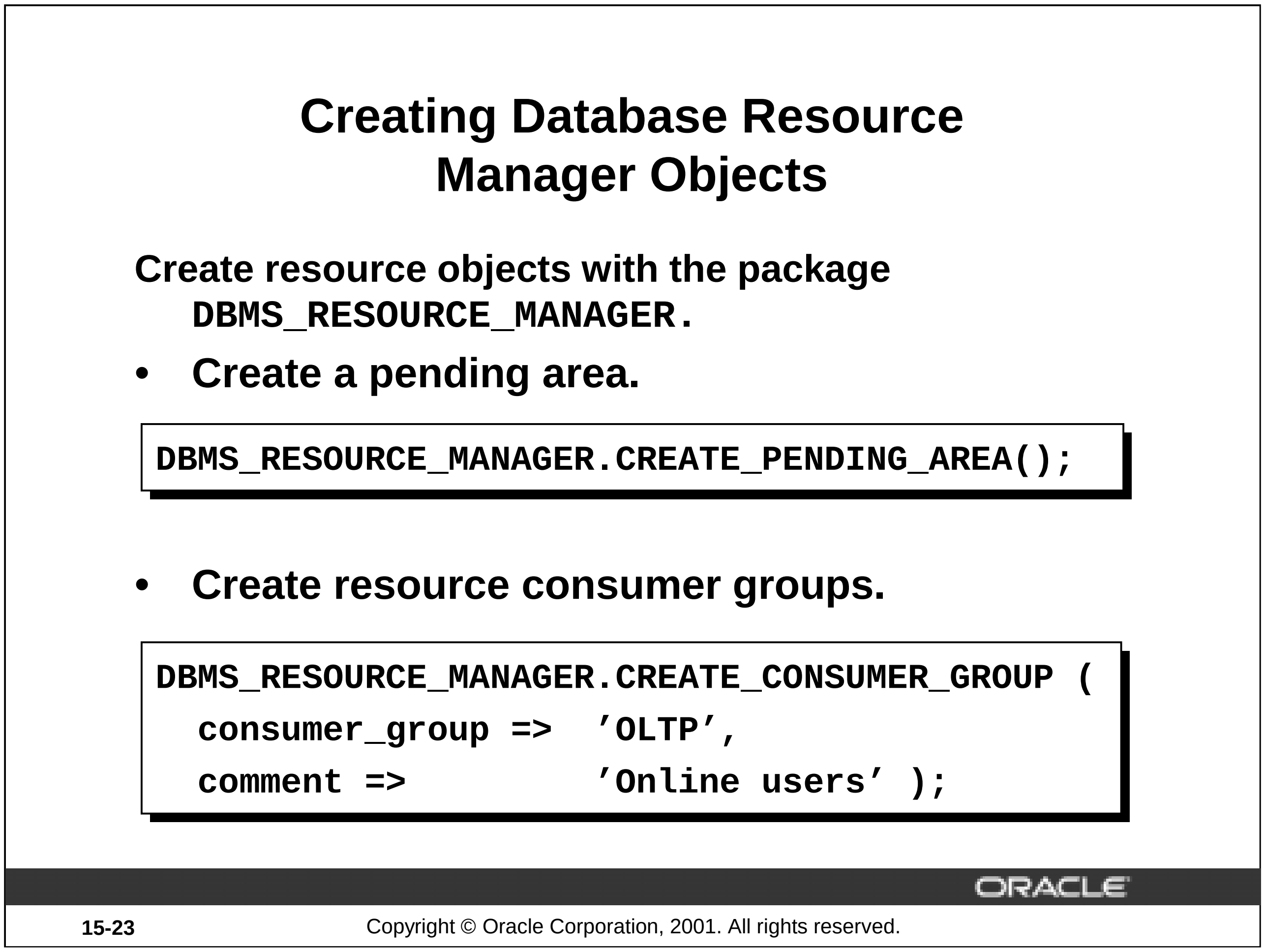
```
DBMS_RESOURCE_MANAGER_PRIVS.
  GRANT_SYSTEM_PRIVILEGE (
    grantee_name => ’SCOTT’,
    privilege_name
      => ’ADMINISTER_RESOURCE_MANAGER’,
    admin_option => FALSE  );
```

Granting Privileges Needed to Administer Resources

The `DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SYSTEM_PRIVILEGE` procedure is used to grant the privilege to a user. For example, to permit the order entry users to manage database resources:

```
DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SYSTEM_PRIVILEGE(
grantee_name => ’OE’,
privilege_name => ’ADMINISTER_RESOURCE_MANAGER’,
admin_option => FALSE );
```

The `privilege_name` parameter defaults to `ADMINISTER_RESOURCE_MANAGER`. The third parameter specifies that OE has been granted the privilege without the `ADMIN OPTION`.



Creating Database Resource Manager Objects

Create resource objects with the package
`DBMS_RESOURCE_MANAGER.`

- Create a pending area.

```
DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
```

- Create resource consumer groups.

```
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP (
  consumer_group =>  'OLTP',
  comment =>         'Online users' );
```

Creating a Pending Area

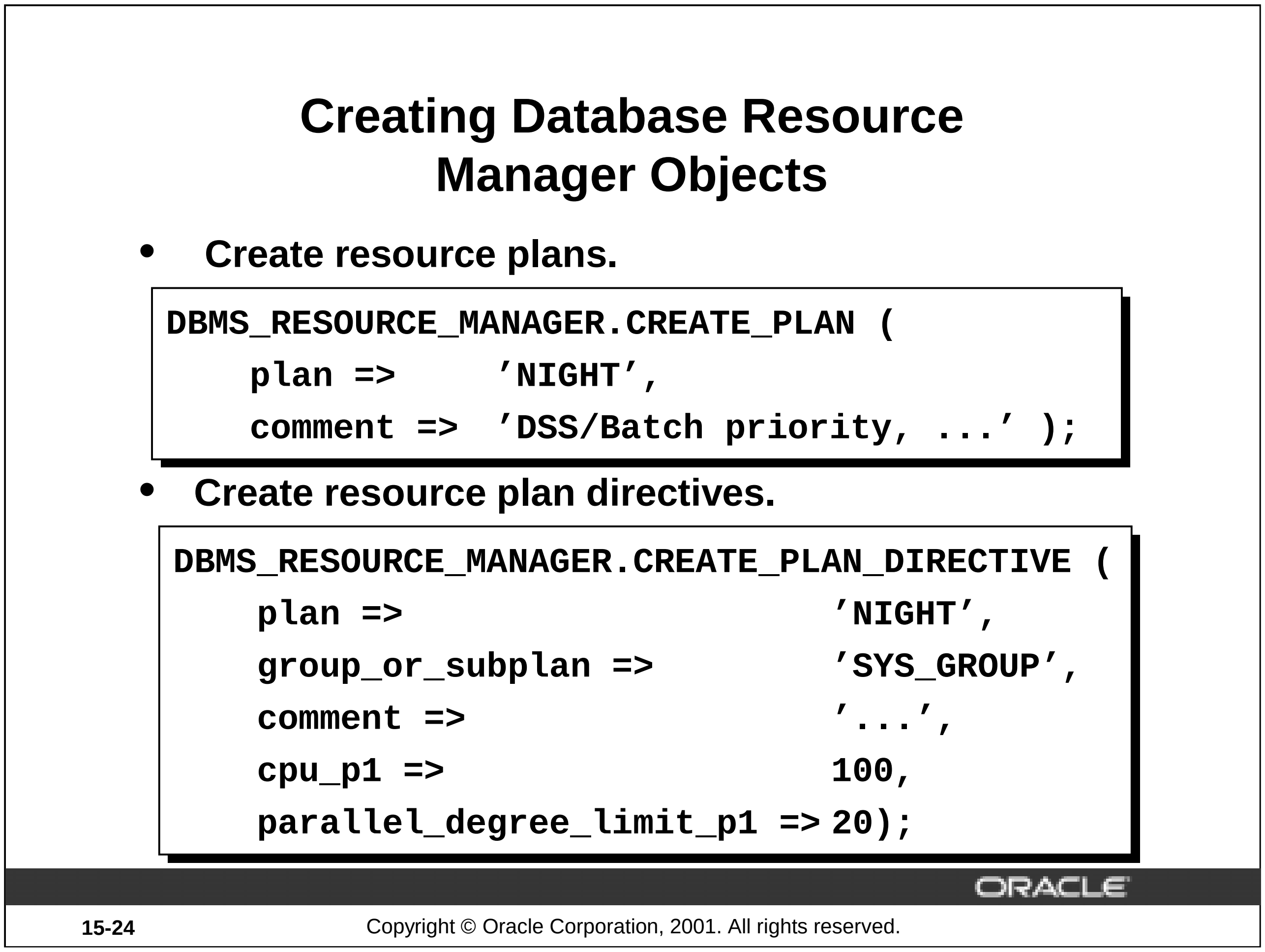
This is a scratch area allowing you to stage your changes and to validate them before they are made active. Only one pending area can be created in an instance at a given point in time. It is created using the procedure, `CREATE_PENDING_AREA`. Views are available for inspecting all active resource plan as well as the pending ones (see at the end of this Lesson).

```
DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
```

Creating Resource Consumer Groups

When a consumer group is defined, it is stored in the pending area. A DBA can specify the name and a comment while defining a consumer group. Procedures are available to modify or delete a consumer group.

```
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP (
consumer_group => 'OLTP', comment => 'Online users');
```



Creating Database Resource Manager Objects

- **Create resource plans.**

```
DBMS_RESOURCE_MANAGER.CREATE_PLAN (
    plan =>    'NIGHT',
    comment => 'DSS/Batch priority, ...' );
```

- **Create resource plan directives.**

```
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
    plan =>                           'NIGHT',
    group_or_subplan =>               'SYS_GROUP',
    comment =>                        '...',
    cpu_p1 =>                         100,
    parallel_degree_limit_p1 => 20);
```

15-24

Creating Resource Plans

When a resource plan is defined, it is stored in the pending area. A DBA can specify the name and a comment while defining a resource plan. Procedures are also available to modify or delete a resource plan. Other arguments like `CPU_MTH`, `MAX_ACTIVE_SESS_TARGET_MTH`, `PARALLEL_DEGREE_LIMIT_MTH` can be specified when creating the plan but they cannot be changed at the moment and you should leave the default values.

```
DBMS_RESOURCE_MANAGER.CREATE_PLAN (
plan => 'NIGHT', comment => 'DSS/Batch priority, ...');
```

Creating Resource Plan Directives

A consumer group or a subplan is associated with a resource plan using the `CREATE_PLAN_DIRECTIVE` procedure. A DBA uses the arguments `cpu_p1, cpu_p2,..., cpu_p8` to distribute CPU resources up to eight levels.

```
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
plan => 'NIGHT', group_or_subplan => 'SYS_GROUP',
comment => '...', cpu_p1 => 100,
parallel_degree_limit_p1 => 20);
```

Creating Database Resource Manager Objects

- **Validate the Pending Area**

```
DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
```

- **Commit the Pending Area**

```
DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
```

Validating the Pending Area

The `VALIDATE_PENDING_AREA` procedure can be used to validate the changes made to the pending area at any time. When validating, the following rules must be adhered to:

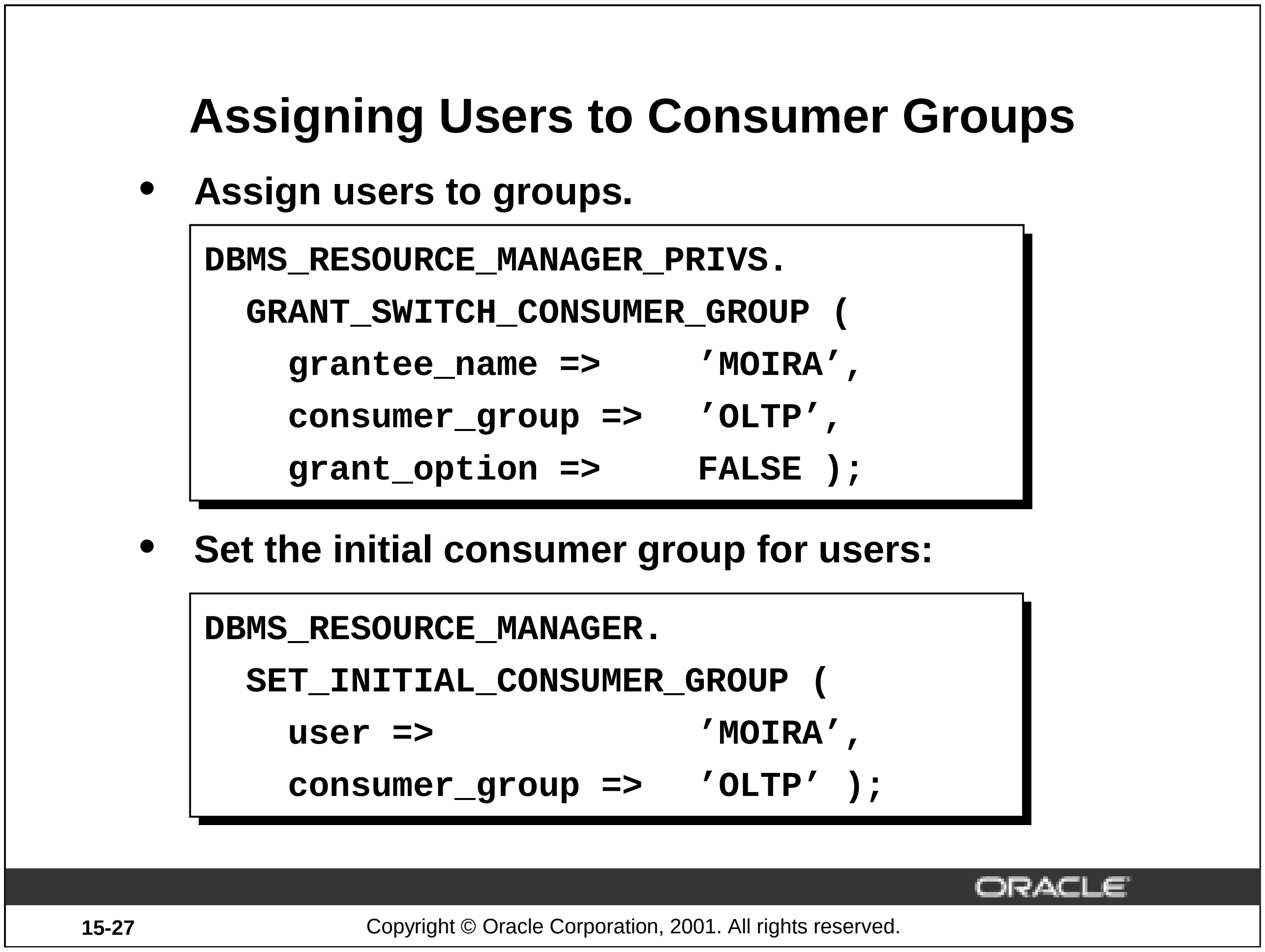
- No plan schema can contain any loops.
- All plan and/or resource consumer groups referred to by plan directives must exist.
- All plans must have plan directives that point to either plans or resource consumer groups.
- All percentages in any given level must not add up to greater than 100 for the EMPHASIS resource allocation method.
- A plan that is currently being used as a top plan by an active instance cannot be deleted.
- The plan directive parameter `PARALLEL_DEGREE_LIMIT_P1` can appear only in plan directives that refer to resource consumer groups (not other resource plans).
- There can be no more than 32 resource consumer groups in any active plan schema. Also, at most, a plan can have 32 children. All leaves of a top plan must be resource consumer groups; at the lowest level in a plan schema the plan directives must refer to consumer groups.
- Plans and resource consumer groups cannot have the same name.
- There must be a plan directive for `OTHER_GROUPS` somewhere in any active plan schema.

Validating the Pending Area (continued)

If any of the rules are violated, the user receives an error. At this point the user may clear all changes using `CLEAR_PENDING_AREA` and reissue the commands, or use the appropriate procedure to correct the errors.

Committing the Pending Area

After you have validated your changes, you can make them active by calling the `SUBMIT_PENDING_AREA` procedure. If successful, this command clears the pending area. If there is a validation error, an exception is raised and the user can make corrections before invoking this procedure again. The submit procedure also performs validation, so you are not forced to call the validate procedure. However, debugging problems is often easier if you incrementally validate your changes especially if you are manipulating complex plans.



Assigning Users to Consumer Groups

- **Assign users to groups.**

```
DBMS_RESOURCE_MANAGER_PRIVS.
  GRANT_SWITCH_CONSUMER_GROUP (
    grantee_name =>     'MOIRA',
    consumer_group =>   'OLTP',
    grant_option =>     FALSE );
```

- **Set the initial consumer group for users:**

```
DBMS_RESOURCE_MANAGER.
  SET_INITIAL_CONSUMER_GROUP (
    user =>             'MOIRA',
    consumer_group =>   'OLTP' );
```

Assigning Users to Groups

Before enabling the Database Resource Manager, users must be assigned to resource consumer groups. The DBMS_RESOURCE_MANAGER_PRIVS package contains the procedure to assign resource consumer groups to users by granting the switch privilege to a user, who can then alter its own consumer group. You do not use a pending area for any of these procedures.

```
DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SWITCH_CONSUMER_GROUP (
grantee_name => 'MOIRA', consumer_group => 'OLTP',
grant_option => FALSE );
```

Setting the Initial Consumer Group for Users

The user's initial consumer group is the one to which any session created by that user initially belongs. If it is not set for a user, the user's initial consumer group will default to DEFAULT_CONSUMER_GROUP. You must directly grant to the user or PUBLIC the *switch* privilege to a consumer group before it can be the user's initial consumer group. The switch privilege cannot come from a role granted to that user.

```
DBMS_RESOURCE_MANAGER.SET_INITIAL_CONSUMER_GROUP (
user => 'MOIRA', consumer_group => 'OLTP' );
```

Setting the Resource Plan for an Instance

Specify the plan to be used by the instance.

- **Specify the RESOURCE_MANAGER_PLAN initialization parameter.**

```
RESOURCE_MANAGER_PLAN=day
```

- **Change the resource plan without shutting down and restarting the instance.**

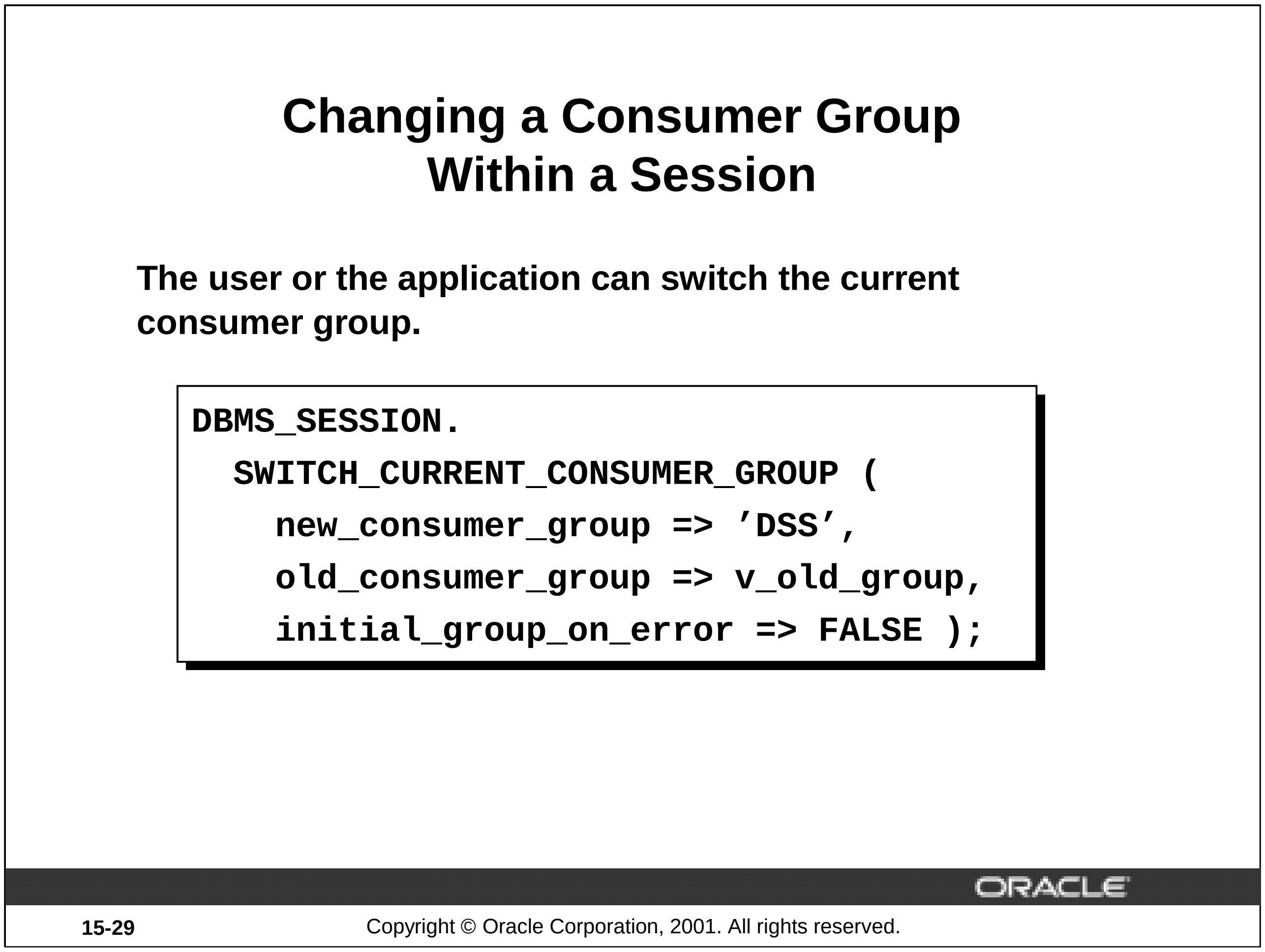
```
ALTER SYSTEM
  SET RESOURCE_MANAGER_PLAN=night;
```

Using the `RESOURCE_MANAGER_PLAN` Initialization Parameter

The plan for an instance can be defined using the `RESOURCE_MANAGER_PLAN` parameter. This parameter specifies the top plan to be used for this instance. If no plan is specified, the Database Resource Manager is not activated for the instance. If the plan specified in the parameter file is not defined in the database, the database cannot be opened and the following error is returned:

```
ORA-07452: specified resource manager plan does not exist
in the data dictionary
```

If this error is encountered, the instance must be shut down and restarted after the parameter is modified to show a correct value. You can also activate, deactivate, or change the current top plan by using the `ALTER SYSTEM` statement. If the resource plan is changed using this command, it takes effect immediately.



Changing a Consumer Group Within a Session

The user or the application can switch the current consumer group.

```
DBMS_SESSION.
  SWITCH_CURRENT_CONSUMER_GROUP (
    new_consumer_group => ’DSS’,
    old_consumer_group => v_old_group,
    initial_group_on_error => FALSE );
```

15-29

Changing the Current Consumer Group

When logged in a session, a user can use the `SWITCH_CURRENT_CONSUMER_GROUP` procedure to switch to another resource consumer group. The new group must be one to which the user has been specifically authorized to switch. If the caller is another procedure, then this procedure enables users to switch to a consumer group for which the owner of that procedure has switch privileges.

```
DBMS_SESSION.SWITCH_CURRENT_CONSUMER_GROUP (
new_consumer_group => ’DSS’,
old_consumer_group => v_old_group,
initial_group_on_error => FALSE );
```

For example, an online application that wants to generate a report at the end of a user session could execute the command shown so that the report runs at a different priority than the rest of the application. The old value is returned to the calling application. If necessary, the consumer group can be switched back to the user’s initial group within the application. The third argument, if TRUE, sets the current consumer group of the invoker to the initial consumer group in the event of an error.

Changing Consumer Groups for Sessions

- **Can be set by DBA for a session**

```
DBMS_RESOURCE_MANAGER.
  SWITCH_CONSUMER_GROUP_FOR_SESS (
    session_id => 7,
    session_serial => 13,
    consumer_group => ’OLTP’);
```

- **Can be set by DBA for all sessions for a user**

```
DBMS_RESOURCE_MANAGER.
  SWITCH_CONSUMER_GROUP_FOR_USER (
    user => ’MOIRA’,
    consumer_group => ’OLTP’);
```

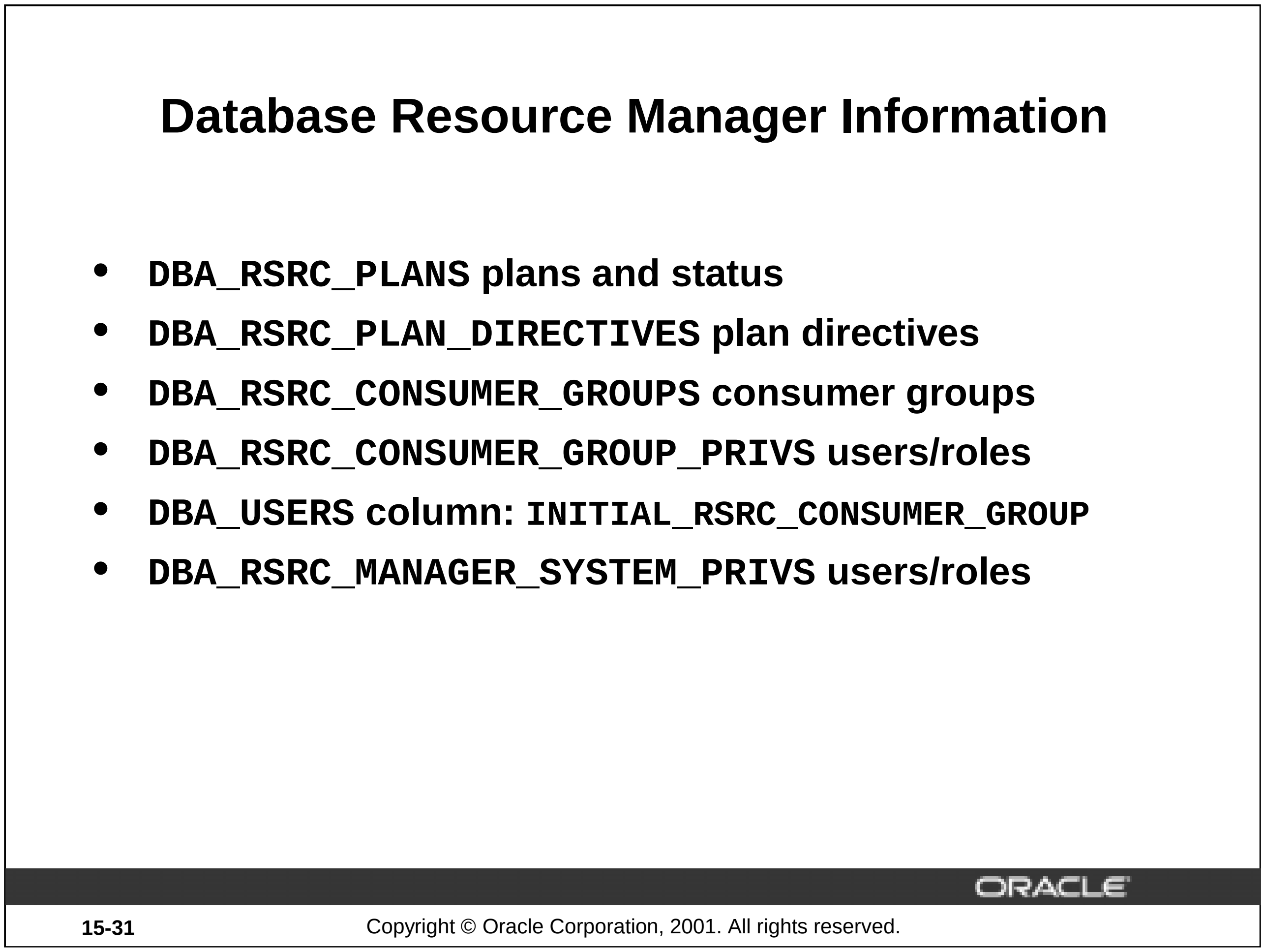
Changing Consumer Groups

A database administrator can switch the consumer group for a session or for a user. These changes take effect immediately. To switch the consumer group for a session, use the SWITCH_CONSUMER_GROUP_FOR_SESS procedure and specify the session ID, serial number, and new consumer group for the session. The user must have been assigned the consumer group that is specified for this operation to succeed. To determine the session ID and serial number, query V$SESSION:

```
SQL> SELECT sid, serial#
  2  FROM v$session
  3  WHERE USERNAME = ’MOIRA’;
SID        SERIAL#
---------- ----------
7          13
```

The SWITCH_CONSUMER_GROUP_FOR_USER procedure provides a convenient method for the database administrator to switch all sessions for a given user that are to be switched into a new consumer group. The username and the new consumer group are the parameters passed to this procedure.

Note that both of these procedures will also change the consumer group of any parallel query slave sessions associated with the coordinator’s session.



Database Resource Manager Information

- **`DBA_RSRC_PLANS` plans and status**
- **`DBA_RSRC_PLAN_DIRECTIVES` plan directives**
- **`DBA_RSRC_CONSUMER_GROUPS` consumer groups**
- **`DBA_RSRC_CONSUMER_GROUP_PRIVS` users/roles**
- **`DBA_USERS` column: `INITIAL_RSRC_CONSUMER_GROUP`**
- **`DBA_RSRC_MANAGER_SYSTEM_PRIVS` users/roles**

Database Resource Manager Information

Several new data dictionary views are available to check the resource plans, consumer groups and plan directives declared in the instance. This section discusses some useful information that can be obtained from these views. For more detailed information about the contents of each of these views refer to *Oracle9i Reference*.

Resource Plans

Use the following query to get information on resource plans defined in the database:

```
• SQL> SELECT plan, num_plan_directives, status, mandatory
• 2 FROM dba_rsrc_plans;
• PLAN           NUM_PLAN_DIRECTIVES STATUS    MAN
• -------------- ------------------- --------- ---
• NIGHT_PLAN                       3 ACTIVE    NO
• SYSTEM_PLAN                      3 ACTIVE    NO
```

A `STATUS` of `ACTIVE` indicates that the plan has been submitted and can be used, while a status of `PENDING` shows that the plan has been created, but is still in the pending area.

If the `MANDATORY` column is assigned the YES value then the plan cannot be deleted.

Resource Plan Directives

Use the following query to retrieve informations on resource plan directives defined in the database:

```
SQL> SELECT plan, group_or_subplan, cpu_p1, cpu_p2, cpu_p3,
2 parallel_degree_limit_p1, status
3 FROM dba_rsrc_plan_directives;
```

```
PLAN         GROUP_OR_SUBPLA  CPU_P1  CPU_P2  CPU_P3 PARALL ST
------------ --------------- ------- ------- ------- ------ --
NIGHT_PLAN   BATCH                70       0       0     20 AC
NIGHT_PLAN   OLTP                 25       0       0      2 AC
NIGHT_PLAN   OTHER_GROUPS          5       0       0      2 AC
SYSTEM_PLAN  SYS_GROUP           100       0       0      0 AC
SYSTEM_PLAN  OTHER_GROUPS          0     100       0      0 AC
SYSTEM_PLAN  LOW_GROUP             0       0     100      0 AC
```

Note that the SYSTEM_PLAN specifies SYS_GROUP at level-1, OTHER_GROUPS at level-2 and LOW_GROUP at level-3 for CPU usage. This setting will simulate a priority system.

Resource Consumer Groups and Privileges

To list all resource consumer groups and the users and roles assigned to them use the following queries:

```
SQL> SELECT *
2 FROM dba_rsrc_consumer_group_privs;
```

```
GRANTEE                    GRANTED_GROUP                  GRA I
-------------------------- ------------------------------ --- -
BRUCE                      BATCH                          NO  N
DAVID                      BATCH                          NO  Y
DAVID                      OLTP                           YES N
JEAN-FRANCOIS              OLTP                           YES Y
PUBLIC                     DEFAULT_CONSUMER_GROUP         YES Y
PUBLIC                     LOW_GROUP                      NO  N
SYSTEM                     SYS_GROUP                      NO  Y
```

GRANTEE	is the user or role receiving the grant
GRANTED_GROUP	is the granted consumer group name
GRANT_OPTION	Whether grant was with the GRANT option
INITIAL_GROUP	Whether consumer group is designated as the default for this user or role

Resource Consumer Groups and Privileges (continued)

```
SQL> SELECT consumer_group, status, mandatory
2 FROM dba_rsrc_consumer_groups;
CONSUMER_GROUP                 STATUS     MAN
------------------------------ ---------- ---
BATCH                          ACTIVE     NO
OLTP                           ACTIVE     NO
OTHER_GROUPS                   ACTIVE     YES
DEFAULT_CONSUMER_GROUP         ACTIVE     YES
SYS_GROUP                      ACTIVE     NO
LOW_GROUP                      ACTIVE     NO
```

Resource Consumer Groups and Privileges

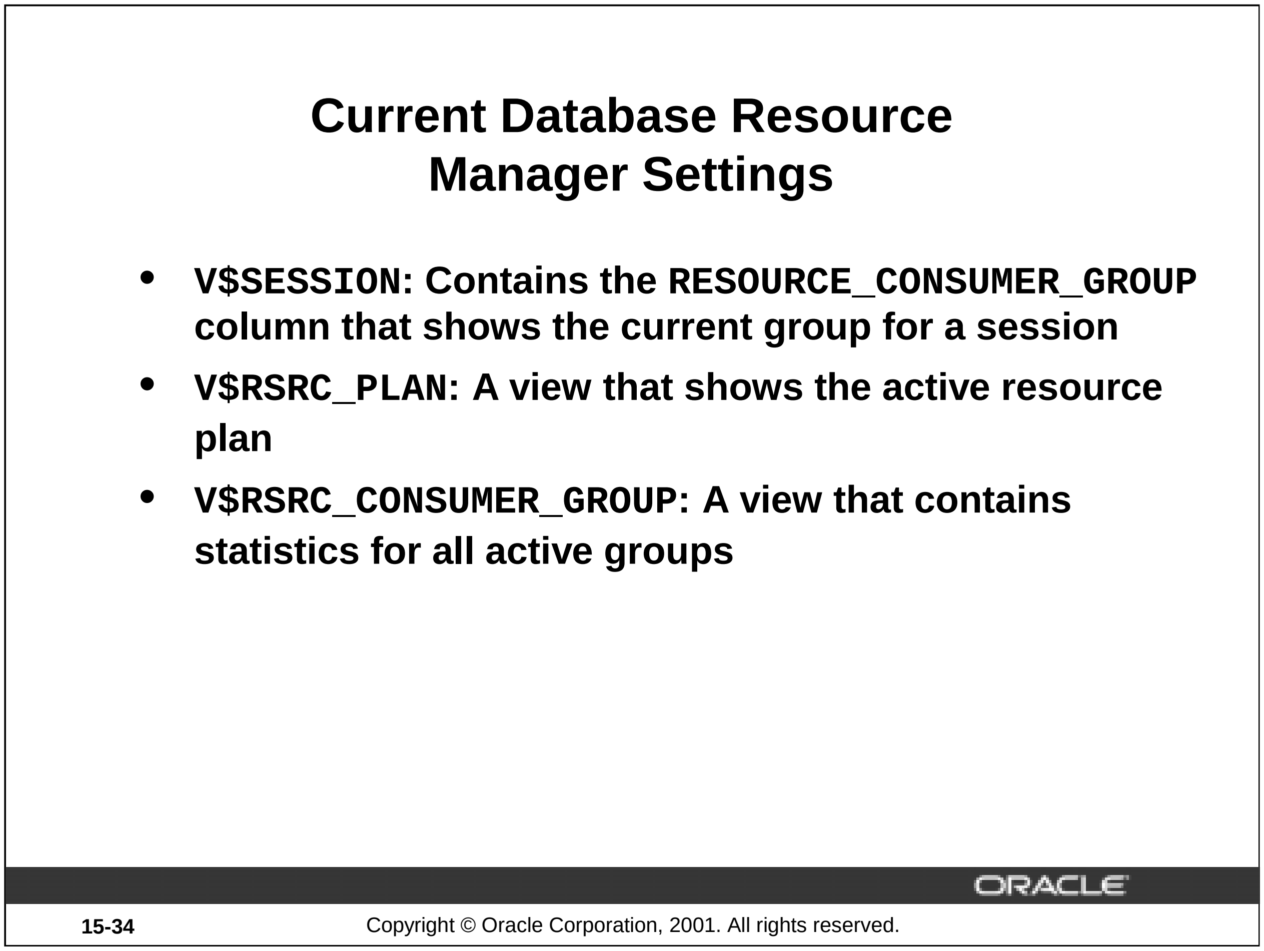
`DBA_RSRC_MANAGER_SYSTEM_PRIVS` lists all the users and roles that have been granted the `ADMINISTER_RESOURCE_MANAGER` system privilege :

```
SQL> select * from dba_rsrc_manager_system_privs;
GRANTEE            PIVILEGE                   ADM
------------------ -------------------------- ---
DBA                ADMINISTER RESOURCE MANAGER YES
EXP_FULL_DATABASE  ADMINISTER RESOURCE MANAGER NO
IMP_FULL_DATABASE  ADMINISTER RESOURCE MANAGER NO
```

Initial Resource Consumer Group

`DBA_USERS` describes all users of the database and the `INITIAL_RSRC_CONSUMER_GROUP` relates to the Database Resource Manager :

```
SQL> select username, initial_rsrc_consumer_group
2> from dba_users;
USERNAME                       INITIAL_RSRC_CONSUMER_GROUP
------------------------------ ------------------------------
SYS                            SYS_GROUP
SYSTEM                         SYS_GROUP
OUTLN                          DEFAULT_CONSUMER_GROUP
```



Current Database Resource Manager Settings

- **`V$SESSION`: Contains the `RESOURCE_CONSUMER_GROUP` column that shows the current group for a session**
- **`V$RSRC_PLAN`: A view that shows the active resource plan**
- **`V$RSRC_CONSUMER_GROUP`: A view that contains statistics for all active groups**

15-34

CPU Utilization

There are at least three different views in the system that can give you information about the CPU utilization inside Oracle*9i* :

- `V$RSRC_CONSUMER_GROUP` shows CPU utilization statistics on a per consumer group basis, if you are running the Oracle Database Resource Manager. This view displays data related to currently active resource consumer groups.
- `V$SYSSTAT` shows Oracle CPU usage for all sessions. The statistic "CPU used by this session" shows the aggregate CPU used by all sessions.
- `V$SESSTAT` shows Oracle CPU usage per session. You can use this view to determine which particular session is using the most CPU.

`V$RSRC_CONSUMER_GROUP`

Here is a quick description of some of its columns :

- `NAME`: Name of the consumer group
- `ACTIVE_SESSIONS`: Number of currently active sessions in this consumer group
- `EXECUTION_WAITERS`: Number of active sessions waiting for a time slice.
- `REQUESTS`: Cumulative number of requests that were executed in this consumer group
- `CPU_WAIT_TIME`: Cumulative amount of time that sessions waited for CPU
- `CONSUMED_CPU_TIME`: Cumulative amount of CPU time consumed by all sessions.

Guidelines

- Tune it all.
- Tune the OS using Oracle statistics.
- Initialization parameters indirectly affect performance.
- Reduce hardware demand through off-loading.

Tuning

Tune the Whole System

To have a well-tuned system, the operating system, the database, and the application must all be tuned. The operating system or database may be performing poorly because of a poorly written application.

For example, an application does many full table scans. The full table scans increase system I/O and have a negative affect on the database buffer cache statistics. The developer tunes the application by adding indexes. The reduction in full table scans causes the database buffer cache to be used more effectively and reduces operating system I/O.

Tune the Operating System Using Oracle Statistics

The system administrator can use information from the database administrator to tune the operating system. For example, the I/O statistics from `V$FILESTAT` can be used to determine which Oracle data files to move to balance I/O.

Summary

In this lesson, you should have learned how to explain the ways in which:

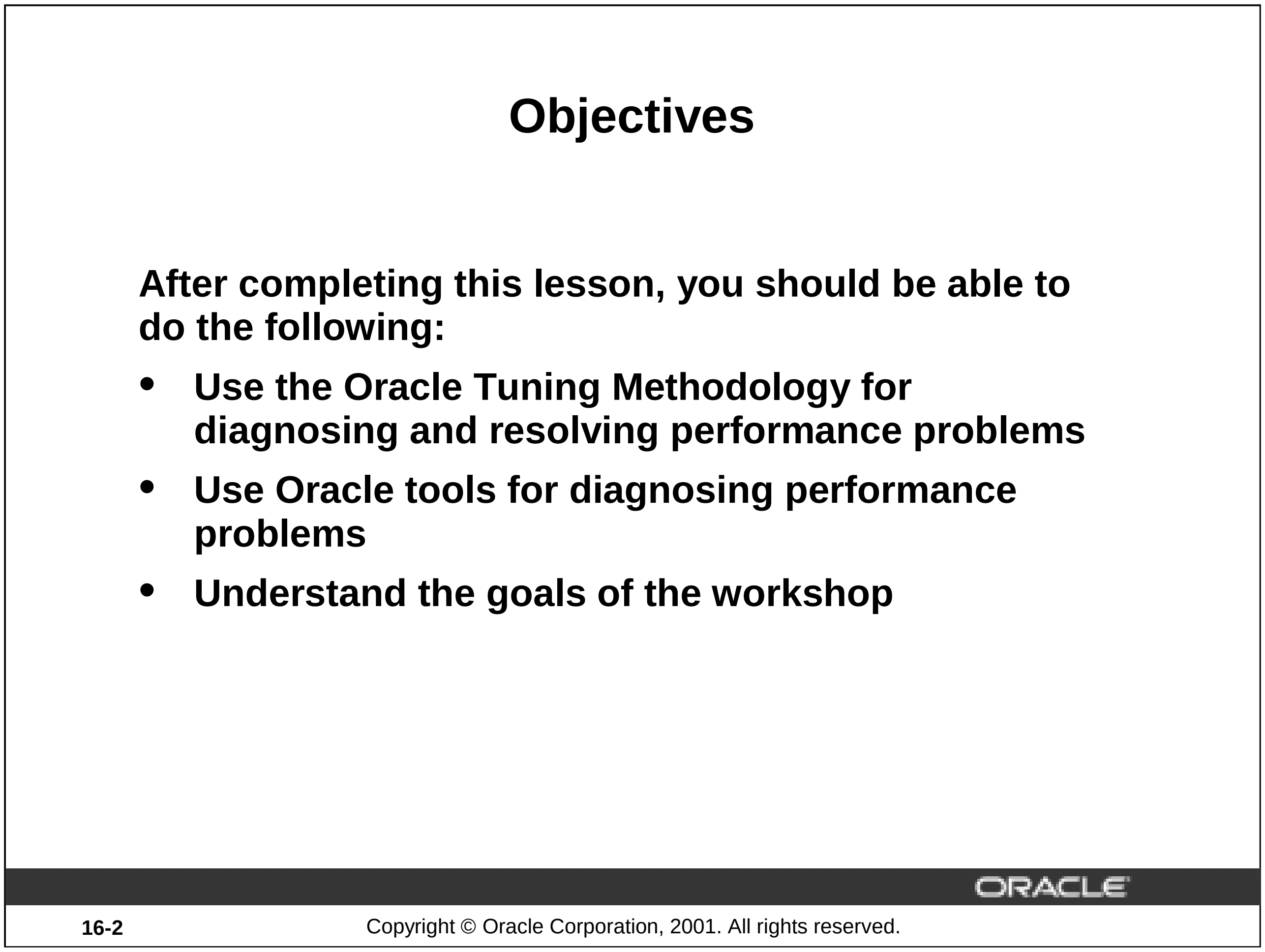
- The system administrator and the DBA tune together
  - OS tuning affects database performance
  - Database tuning affects OS performance
- Resource Manager Groups are set up
- Users are assigned to different plans within a group

15-36

16

Workshop Overview

ORACLE



Objectives

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

- Use the Oracle Tuning Methodology for diagnosing and resolving performance problems
- Use Oracle tools for diagnosing performance problems
- Understand the goals of the workshop

Approach to Workshop

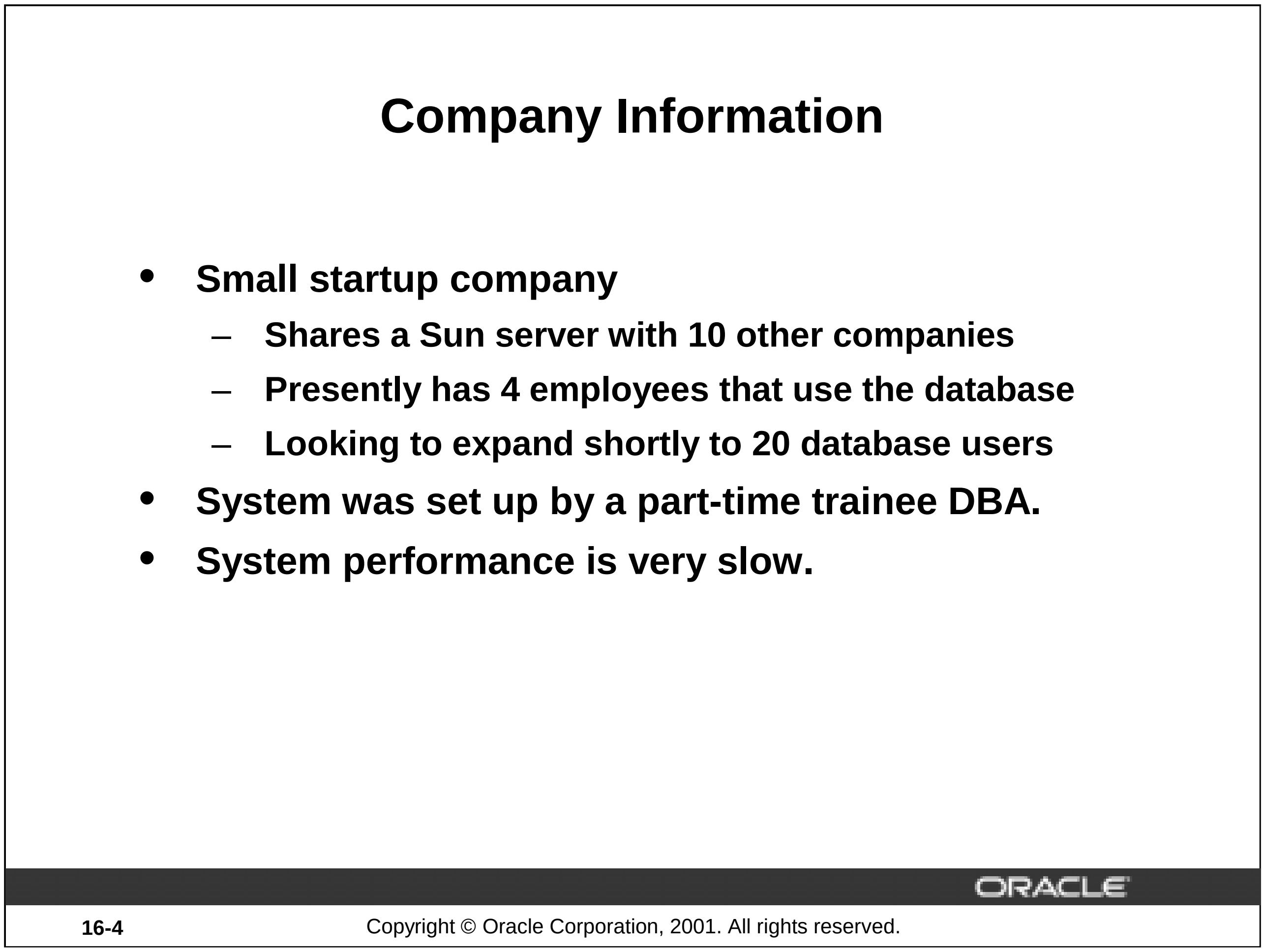
- **Group-oriented and interactive**
- **Intensive hands-on diagnosis and problem resolution**
- **Proactive participant involvement**

Group-Oriented and Interactive

The workshop is structured to allow groups of individuals to work together to perform tuning diagnostics and resolution. Each group is encouraged to share its tuning diagnostic and resolution approach with other groups in the class.

Intensive Hands-On Diagnosis and Problem Resolution

The intent is to provide you with as much hands-on experience as possible to diagnose and work through a performance tuning methodology, diagnosis, and resolution. The experience and knowledge gained from the first four days of the Oracle9*i* : Performance Tuning course play a major role in successfully completing the workshop.



Company Information

- **Small startup company**
  - **Shares a Sun server with 10 other companies**
  - **Presently has 4 employees that use the database**
  - **Looking to expand shortly to 20 database users**
- **System was set up by a part-time trainee DBA.**
- **System performance is very slow.**

Company Information

Present Situation

The company has 2 OLTP type users, and 2 DSS type. The system was set up by a trainee DBA, and though it works, the performance is very slow. The company rents space on a Sun server which it shares with 10 other companies. Due to this there is a requirement that resources used be kept to a minimum.

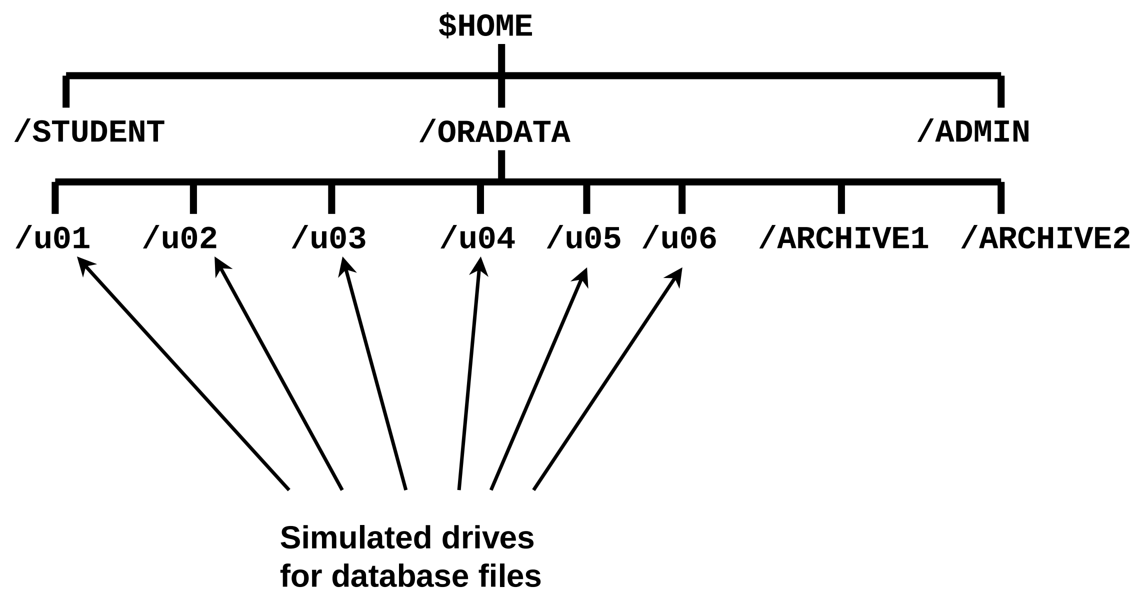
Future Goals

The company is expanding. The goal is to have twenty concurrent database users. You have been invited in to get the system ready for the new workload. It is expected that there will be 10 of each type of user.

After collecting whatever statistics you feel necessary. Implement whatever database changes your investigation determines would improve the situation. For example, what init.ora parameter values you would set, could some tables do with an index, and so on.

As part of your assignment is to recommend any extra hardware that might be required.

Workshop Configuration



Directories

Each lab account is set up in the following manner:

- Each user account has its own $HOME directory.
- Six directories under $HOME/ORADATA are used to simulate multiple physical devices (u01, u02, u03, u04, u05, and u06).
- The ADMIN directory is used for instance-specific trace files.
- The STUDENT directory contains lab and workshop files.

Workshop Database Configuration

The database consists of a sample schema.

- **Users log in as `oltp`, or `dss`, depending on the nature of their work.**
- **End users have access to sample schema objects.**
- **There are seven tablespaces: `system`, `undotbs`, `temp`, `users`, `indx`, `sample`, `tools`.**
- **The DBA account is `system/manager`.**
- **The sys account is `sys/change_on_install`.**

Database Configuration

Use the Oracle data dictionary views to get a complete picture of the database configuration.

Setup for Statspack

Ensure that the job scheduler is set to collect statistics every 10 minutes. This is done by connecting as the user perfstat and executing the following statement:

```
SQL> select job, next_date, next_sec
    2   from user_jobs;
```

Workshop Procedure

- Choose a scenario.
- Run the workload generator.
- Create a STATSPACK report.
- Determine what changes should take place.
- Implement the changes.
- Return to the second step to check that the changes have made an improvement.
- When the changes have improved performance, choose another scenario.

Choosing a Scenario

- Change directory into `$HOME/STUDENT/WKSHOP.`
- Execute the `wksh` script.
- Select the scenario number desired.
- Script then makes the relevant changes.
- Leave database up and running.

16-8

Executing the Script

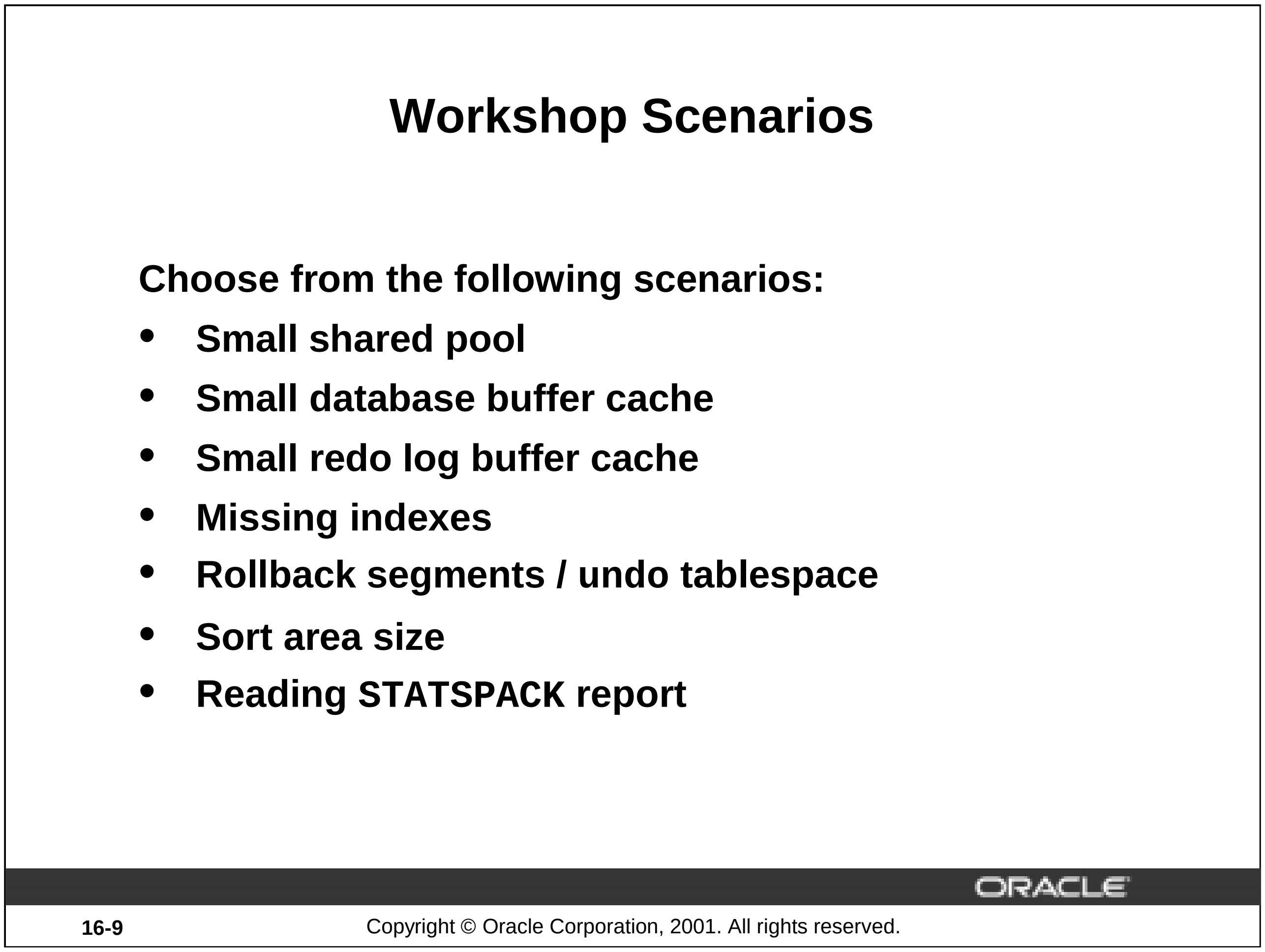
In order to choose the scenario required, and set up the database appropriately run the wksh script that is located in the directory $HOME/STUDENT/WKSHOP.

```
$ cd $HOME/STUDENT/WKSHOP
$ ./wksh
```

Follow the prompts from this point.

Once the script has completed the database will be left in an active condition, however, there are problems with the instance as per the scenario chosen. It is now required to resolve these issues.

When resizing memory components, remember not to consume more memory than is actually required in order to meet the objects given. For example, there is little to be gained in resizing the shared pool to 500Mb. The purpose of the workshop is to be as realistic as possible.



Workshop Scenarios

Choose from the following scenarios:

- Small shared pool
- Small database buffer cache
- Small redo log buffer cache
- Missing indexes
- Rollback segments / undo tablespace
- Sort area size
- Reading STATSPACK report

16-9

Workshop Scenario 5

After completing this scenario it is required to change the database back into Automatic Managed Undo, this can be performed manually, or by running option 0 in the script wksh

Workshop Scenario 7

This scenario is different to the others in that it does not have an option to run with the script wksh. Instead this option invites you to examine an actual report generated by Statspack.

Collecting Information

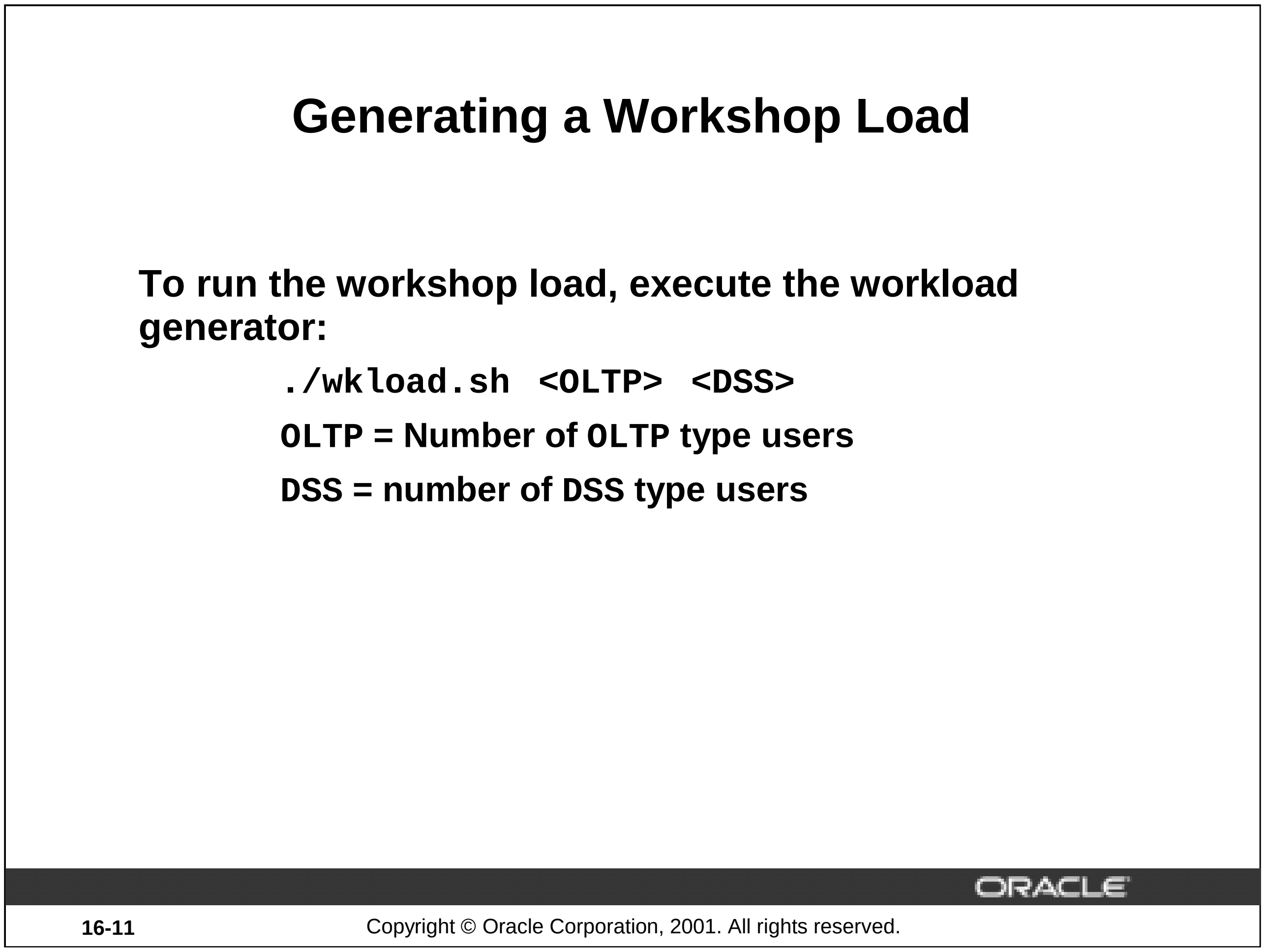
To perform a physical investigation of your workshop database environment, collect statistics using:

- **V$ dynamic performance views**
- **Data dictionary views**
- **Table statistics**
- **STATSPACK**

Physical Investigation

Perform a physical investigation of your workshop database environment. Some areas that you may want to focus on are listed below as a guide. Remember to use the tools that are available to you within the Oracle environment, such as the V$ dynamic performance views, data dictionary views, table statistics, STATSPACK, and so forth. Use the statistics and ratios presented during the first four days of the class to analyze the baseline. Depending on the scenario used, there are a number of statistics in the report file that are contrary to a well-tuned database.

Note: While the workload generator is executing, you can use available tools to monitor database performance. Also ensure that you are in a writable directory when you are run the spreport.sql script.



Generating a Workshop Load

To run the workshop load, execute the workload generator:

```
./wkload.sh <OLTP> <DSS>
```

`OLTP` = Number of `OLTP` type users

`DSS` = number of `DSS` type users

Generating a Workshop Load

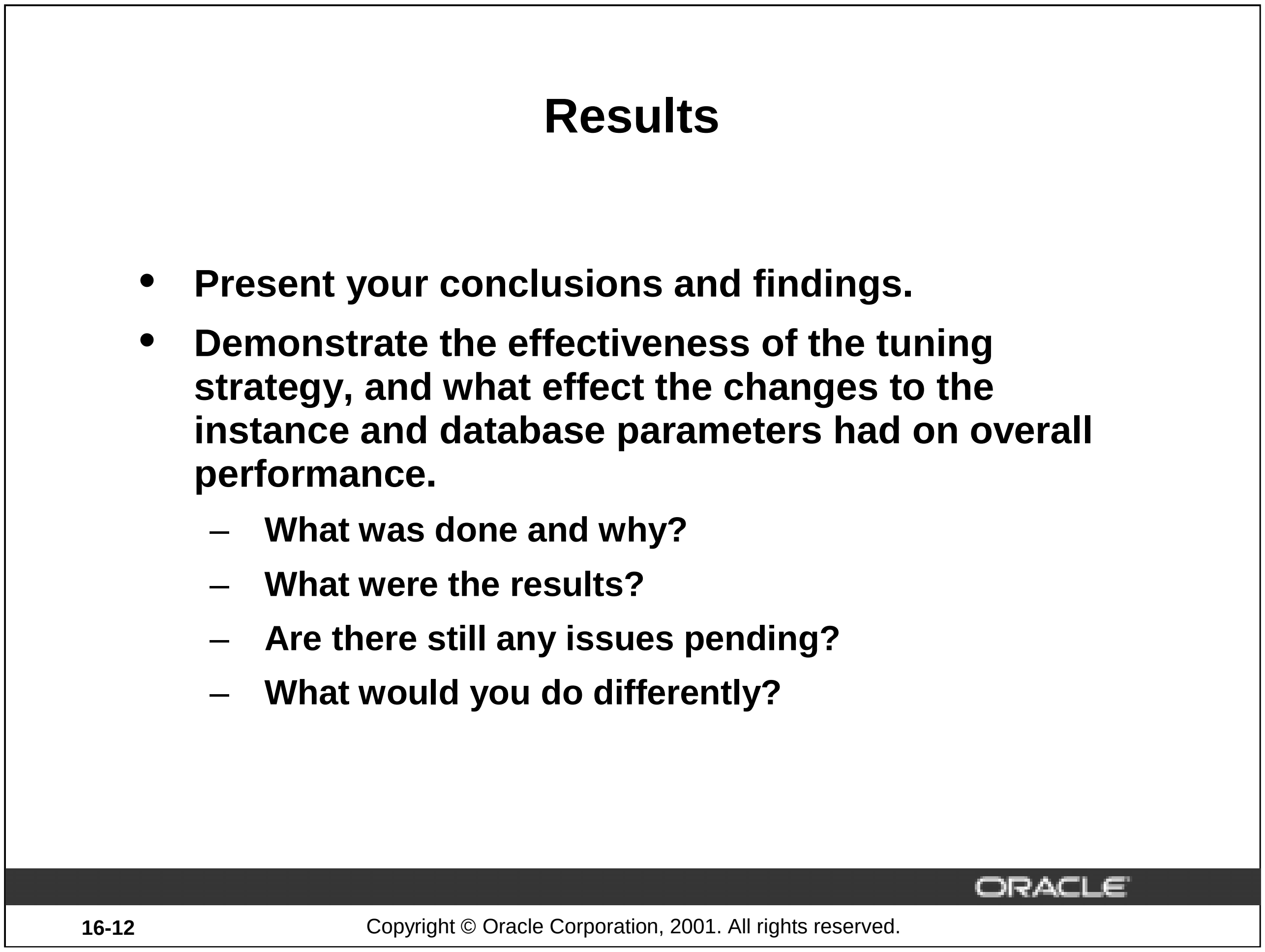
Start the workload generator, and note the time.

- In the $HOME/STUDENT/WKSHOP directory find the script wkload.sh. This script is called with two parameters, number of user for OLTP, and DSS, as per the following example:

```
$ cd $HOME/STUDENT/WKSHOP
$ ./wkload.sh   8   6
```

  Where 8 is the number of OLTP users and 6 the number of DSS users

- Give time for some statistics to be generated (a period of at least 10 minutes should be given). When a number of snapshots have been collected by STATSPACK run the script spreport.sql. For the time period, choose a begin and end time within the period that the workload generator was running. Remember that the closer the snapshot is to the database startup, the more that event will effect the statistics collected.
- Name the report in a manner associated with the scenario, ie for example, for scenario 1 use reportscn1.txt, for Scenario 5 use reportscn5.txt, and so on.
- Look for the generated report.



Results

- **Present your conclusions and findings.**
- **Demonstrate the effectiveness of the tuning strategy, and what effect the changes to the instance and database parameters had on overall performance.**
  - **What was done and why?**
  - **What were the results?**
  - **Are there still any issues pending?**
  - **What would you do differently?**

16-12

Conclusions

Each group presents its conclusions and findings to the rest of class. The intent is to demonstrate the effectiveness of the tuning strategy and show what effect the modifications to the instance and database parameters had on overall performance. Include the following items in your presentation:

- What was done?
- What was the justification?
- What were the results?
- Are there any pending issues?
- What would you do differently?

Summary

In this lesson, you should have learned how to:

- Follow the Oracle Tuning methodology:
  1. Collect and review statistics.
  2. List the objectives for enhanced performance before modifications.
  3. Modify the instance and the database.
  4. Re-collect and review new statistics.
  5. Compare the new results with the objectives.
- Implement Oracle architectural options for enhancing performance