Oracle9*i* Database Performance Tuning

Student Guide Vol 2

D11299GC11
Production 1.1
December 2001
D34122

ORACLE®

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
David Austin
Howard Bradley
Howard Ostrow
Janet Stern
Lilian Hobbs
Maria Senise
Roderick Manalac
Sander Rekveld
Scott Gossett

Contents

1 Overview of Oracle 9i 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
Oracle Wait Events 2-18
The V$EVENT_NAME View 2-19
Statistics Event Views 2-20
The V$SYSTEM_EVENT View 2-21
The V$SESSION_WAIT View 2-23
STATSPACK 2-25
STATSPACK Output 2-27
UTLBSTAT and UTLESTAT Utilities 2-30
Enterprise Manager Console 2-31
Performance Manager 2-32
Overview of Oracle Expert Tuning Methodology 2-34
Tuning Using Oracle Expert 2-35
DBA Developed Tools 2-37
Summary 2-38

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
Library Cache Guidelines STATSPACK Report 3-13
Invalidations 3-14
Sizing the Library Cache 3-16
Cached Execution Plans 3-17
View to Support Cached Execution Plans 3-18
V$SQL Support For Cached Execution Plans 3-19
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
Tuning The Data Dictionary Cache 3-32
Diagnostic Tools for Tuning the Data Dictionary Cache 3-33
Measuring the Dictionary Cache Statistics 3-34
Tuning the Data Dictionary Cache 3-35
Guidelines: Dictionary Cache Misses 3-36
UGA and Oracle Shared Server 3-37
Sizing the User Global Area 3-38
Performance Manager: Shared Pool Statistics 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 Oracle9i 4-5
Dynamic SGA Feature in Oracle9i 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 4-19
Diagnostic Tools 4-21
Measuring the Cache Hit Ratio 4-22
Guidelines for Using the Cache Hit Ratio 4-23
Buffer Cache Hit Ratio Isn't Everything 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 Buffer Cache Performance Indicators 4-39
Other Buffer Cache Performance Indicators (Cont.) 4-40
Performance Manager 4-42
Free Lists 4-43
Diagnosing Free List Contention 4-44
Resolving Free List Contention 4-45
Automatic Segment Space Management 4-47
Auto-management of Free Space 4-48
Multiple I/O Slaves 4-49
Multiple DBWn Processes 4-50
Tuning DBWn I/O 4-51
Summary 4-52

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

Performance Manager 5-8
Redo Log Buffer Tuning Guidelines 5-9
Reducing Redo Operations 5-11
Monitoring Java Pool Memory 5-13
Sizing the SGA for Java 5-14
Summary 5-15

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
Performance Manager: I/O Statistics 6-9
I/O Statistics 6-10
File Striping 6-11
Tuning Full Table Scan Operations 6-13
Table Scan Statistics 6-15
Monitoring Full Table Scan Operations 6-16
Checkpoints 6-18
Performance Manager: Response Time 6-20
Regulating the Checkpoint Queue 6-21
Defining and Monitoring FASTSTART Checkpointing 6-22
Redo Log Groups and Members 6-23
Online Redo Log File Configuration 6-24
Archive Log File Configuration 6-26
Diagnostic Tools 6-27
Summary 6-28

7 Optimizing Sort Operations
Objectives 7-2
The Sorting Process 7-3
Sort Area and Parameters 7-4
New Sort Area Parameters 7-8
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
Performance Manager: Sorts 7-19
Monitoring Temporary Tablespaces 7-20
Temporary Tablespace Configuration 7-21
Summary 7-23

8 Diagnosing Contention for Latches
Objectives 8-2
Purpose of Latches 8-4
Latch Request Types 8-6
Latch Contention 8-7
Performance Manager: Latches 8-9
Reducing Contention for Latches 8-10
Important Latches for the DBA 8-11
Shared Pool and Library Cache Latches 8-13
Summary 8-14

9 Tuning Undo Segments
Objectives 9-2
Automatic Undo Management in Oracle9i 9-3
Tablespace for Automatic Undo Management 9-4
Altering an Undo Tablespace 9-5
Switching Undo Tablespaces 9-6
Dropping an Undo Tablespace 9-7
Setting UNDO_RETENTION 9-8
Other Parameters for Automatic Undo Management 9-10
Monitoring Automatic Undo Management 9-12
Using V$UNDOSTAT 9-13
Performance Manager: Rollback/Undo 9-14
Rollback Segments: Usage 9-15
Rollback Segment Activity 9-16
Rollback Segment Header Activity 9-17
Growth of Rollback Segments 9-18
Tuning the Manually Managed Rollback Segments 9-19
Diagnostic Tools 9-20
Diagnosing Contention for Manual Rollback Segment Header 9-21
Guidelines: Number of Manual Rollback Segments (RBSs) 9-23
Guidelines: Sizing Manual Rollback Segments 9-25
Sizing Transaction Rollback Data 9-26
Using Less Rollback Per Transaction 9-30
Using Less Rollback 9-31
Possible Problems Caused by Small Rollback Segments 9-32
Summary 9-33

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
Performance Manager: Locks 10-22
Deadlocks 10-23
Summary 10-26

11 Tuning the Oracle Shared Server
Objectives 11-2
Overview 11-3
Oracle Shared Server Characteristics 11-4
Monitoring Dispatchers 11-5
Monitoring Dispatchers 11-6
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 SQL Statement Tuning
Objectives 12-2
Optimizer Modes 12-3
Setting the Optimizer Mode 12-4
Optimizer Plan Stability 12-6
Plan Equivalence 12-7
Creating Stored Outlines 12-8
Using Stored Outlines 12-9
Editing Stored Outlines 12-11
Maintaining Stored Outlines 12-13
Enterprise Manager: Maintaining Stored Outlines 12-14
Using Hints in a SQL Statement 12-15
Overview of Diagnostic Tools 12-16
SQL Reports in STATSPACK 12-17
Performance Manager: Top SQL 12-18
EXPLAIN PLAN 12-19
Using SQL Trace and TKPROF 12-20
Enabling and Disabling SQL Trace 12-22
Formatting the Trace File with TKPROF 12-23

TKPROF Statistics 12-25
SQL*Plus AUTOTRACE 12-26
Managing Statistics 12-27
Table Statistics 12-29
Index Statistics 12-30
Index Tuning Wizard 12-31
Column Statistics 12-32
Histograms 12-33
Generating Histogram Statistics 12-34
Gathering Statistic Estimates 12-35
Automatic Statistic Collecting 12-37
Optimizer Cost Model 12-38
Gathering System Statistics 12-40
Gathering System Statistics Example 12-41
Copying Statistics Between Databases 12-43
Example: Copying Statistics 12-44
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
Migration and Chaining 13-24
Detecting Migration and Chaining 13-25
Selecting Migrated Rows 13-26
Eliminating Migrated Rows 13-27

Index Reorganization 13-29
Monitoring Index Space 13-30
Deciding Whether to Rebuild or Coalesce an Index 13-31
Monitoring Index Usage 13-33
Identifying Unused Indexes 13-34
Summary 13-35

14 Application Tuning
Objectives 14-2
The Role of the Database Administrator 14-3
Data Storage Structures 14-4
Selecting the Physical Structure 14-5
Data Access Methods 14-7
Clusters 14-8
Cluster Types 14-9
Situations Where Clusters Are Useful 14-10
B-Tree Indexes 14-11
Compressed Indexes 14-13
Bitmap Indexes 14-14
Creating and Maintaining Bitmap Indexes 14-16
B-Tree Indexes and Bitmap Indexes 14-17
Reverse Key Index 14-18
Creating Reverse Key Indexes 14-19
Enterprise Manager: Index Management 14-20
Index-Organized Tables 14-21
Index-Organized Tables and Heap Tables 14-22
Creating Index-Organized Tables 14-23
IOT Row Overflow 14-24
IOT Dictionary Views 14-25
Using a Mapping Table 14-26
Maintaining a Mapping Table 14-27
Partitioning Methods 14-28
List Partitioning Example 14-33
Partitioned Indexes for Scalable Access 14-34
Statistics Collection for Partitioned Objects 14-39
ANALYZE Statement 14-42
Materialized Views 14-44
Creating Materialized Views 14-45
Refreshing Materialized Views 14-46
Materialized Views: Manual Refreshing 14-48
Query Rewrites 14-49
Materialized Views and Query Rewrites: Example 14-51
Enabling and Controlling Query Rewrites 14-53
Disabling Query Rewrites: Example 14-55

DBMS_MVIEW Package 14-56
OLTP Systems 14-57
OLTP Requirements 14-58
OLTP Application Issues 14-60
Decision Support Systems (Data Warehouses) 14-61
Data Warehouse Requirements 14-62
Data Warehouse Application Issues 14-64
Hybrid Systems 14-65
Summary 14-66

15 Tuning the Operating System and Using Resource Manager
Objectives 15-2
Objectives 15-3
System Architectures 15-4
Virtual and Physical Memory 15-5
Tuning Memory 15-7
Understanding Different I/O System Calls 15-9
CPU Tuning 15-11
Overview of Database Resource Manager 15-14
Database Resource Management Concepts 15-15
Resource Allocation Methods 15-16
The Original Plan: SYSTEM_PLAN 15-18
Using Subplans 15-19
Administering the Database Resource Manager 15-20
Enterprise Manager: Resource Manager 15-22
Assigning the Resource Manager Privilege 15-23
Creating Database Resource Manager Objects 15-24
Active Session Pool 15-26
Active Session Pool Mechanism 15-27
Active Session Pool Parameters 15-28
Setting the Active Session Pool 15-29
Maximum Estimated Execution Time 15-30
Automatic Consumer Group Switching 15-31
Undo Quota 15-32
Creating Database Resource Manager Objects 15-33
Assigning Users to Consumer Groups 15-35
Setting the Resource Plan for an Instance 15-36
Changing a Consumer Group Within a Session 15-37
Changing Consumer Groups for Sessions 15-38
Database Resource Manager Information 15-39
Current Database Resource Manager Settings 15-42
Guidelines 15-43
Summary 15-44

16 Workshop Overview
Objectives 16-2
Approach to Workshop 16-3
Company Information 16-4
Workshop Configuration 16-5
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

Appendix A: Practice Solutions Using SQL Plus

Appendix B: Practice Solutions Using Enterprise Manager

Appendix C: Tuning Workshop

Appendix D: Example of STATSPACK Report

Appendix E: Redundant Arrays of Inexpensive Disks Technology (RAID)

13

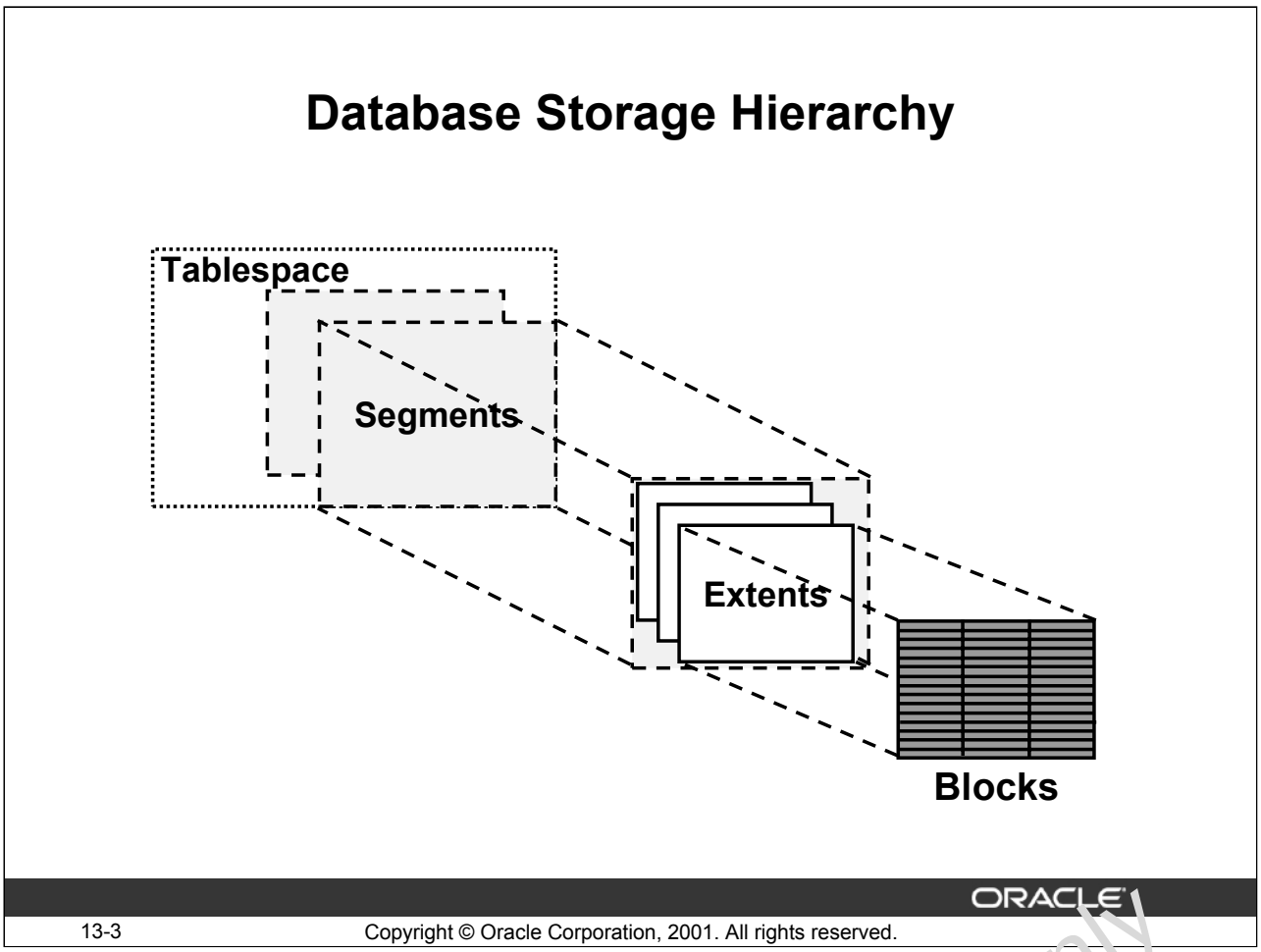
Using Oracle Blocks Efficiently

ORACLE

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**



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.
```


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 extend dynamically 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 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**
  - **Enable you to read the entire extent map with a single I/O operation**
- **Cons:**
  - **Free space may not be available**
  - **Unused space**

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.
- The performance of searches using an index is not affected by the index having one extent or multiple extents.

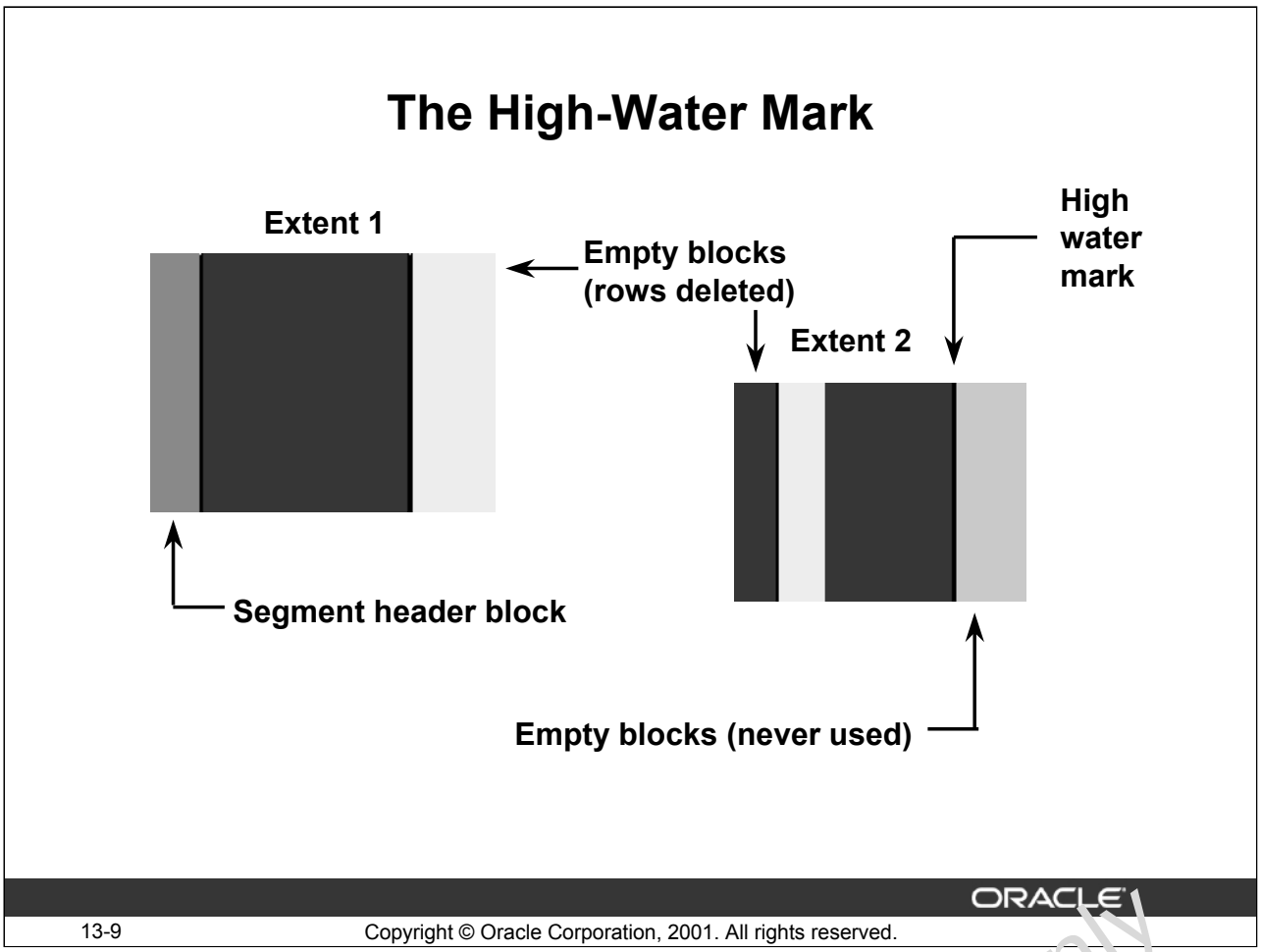
Advantages of Large Extents (continued)

- 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 a 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

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 the 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 using 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
```

ORACLE

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;
/
```

ORACLE

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.

ORACLE

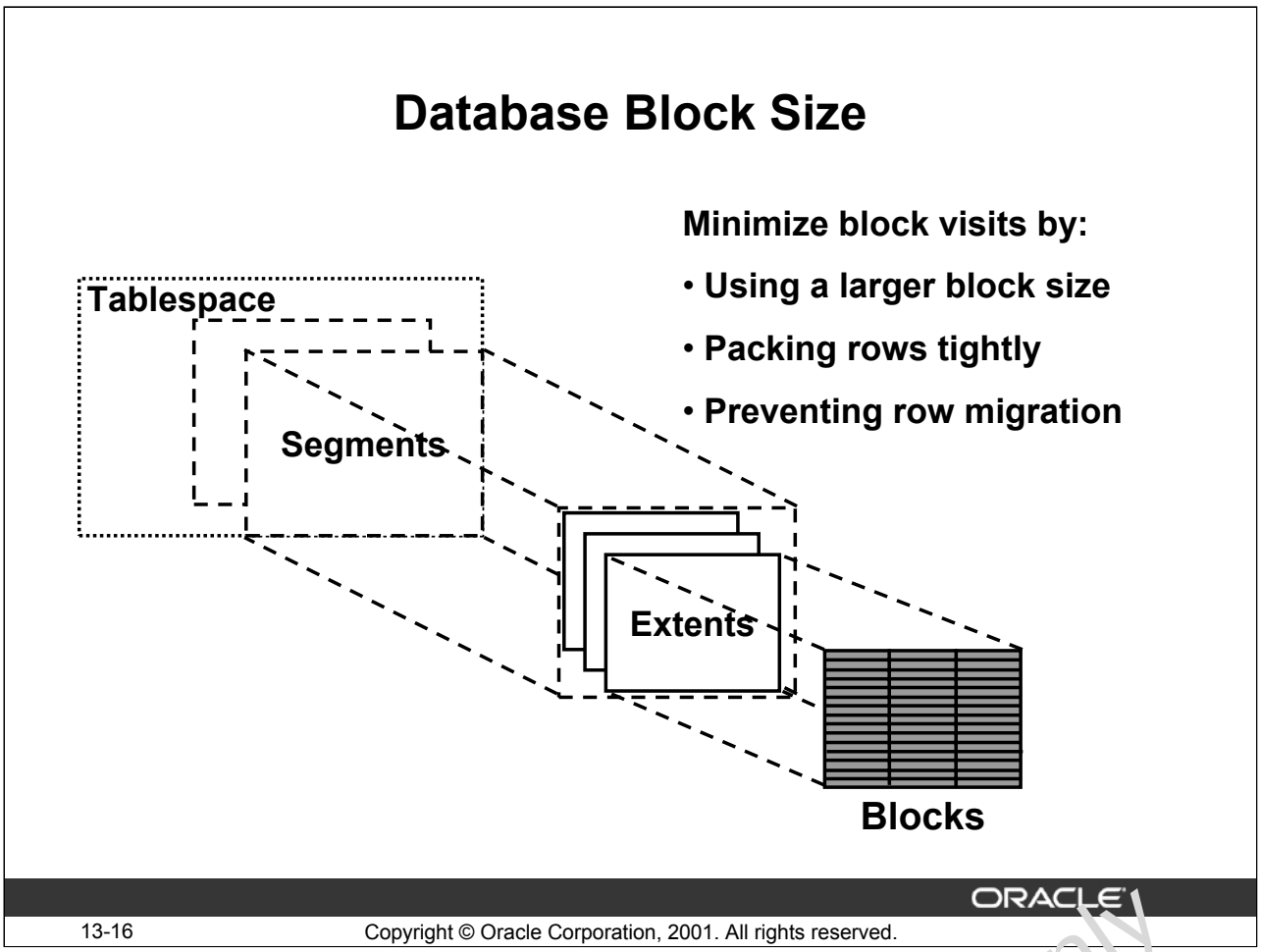
Dropping or Truncating the Table

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.



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**

ORACLE

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**
  - **Can 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 can cause additional I/Os.
- This can 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**

Large Oracle Blocks

Advantages

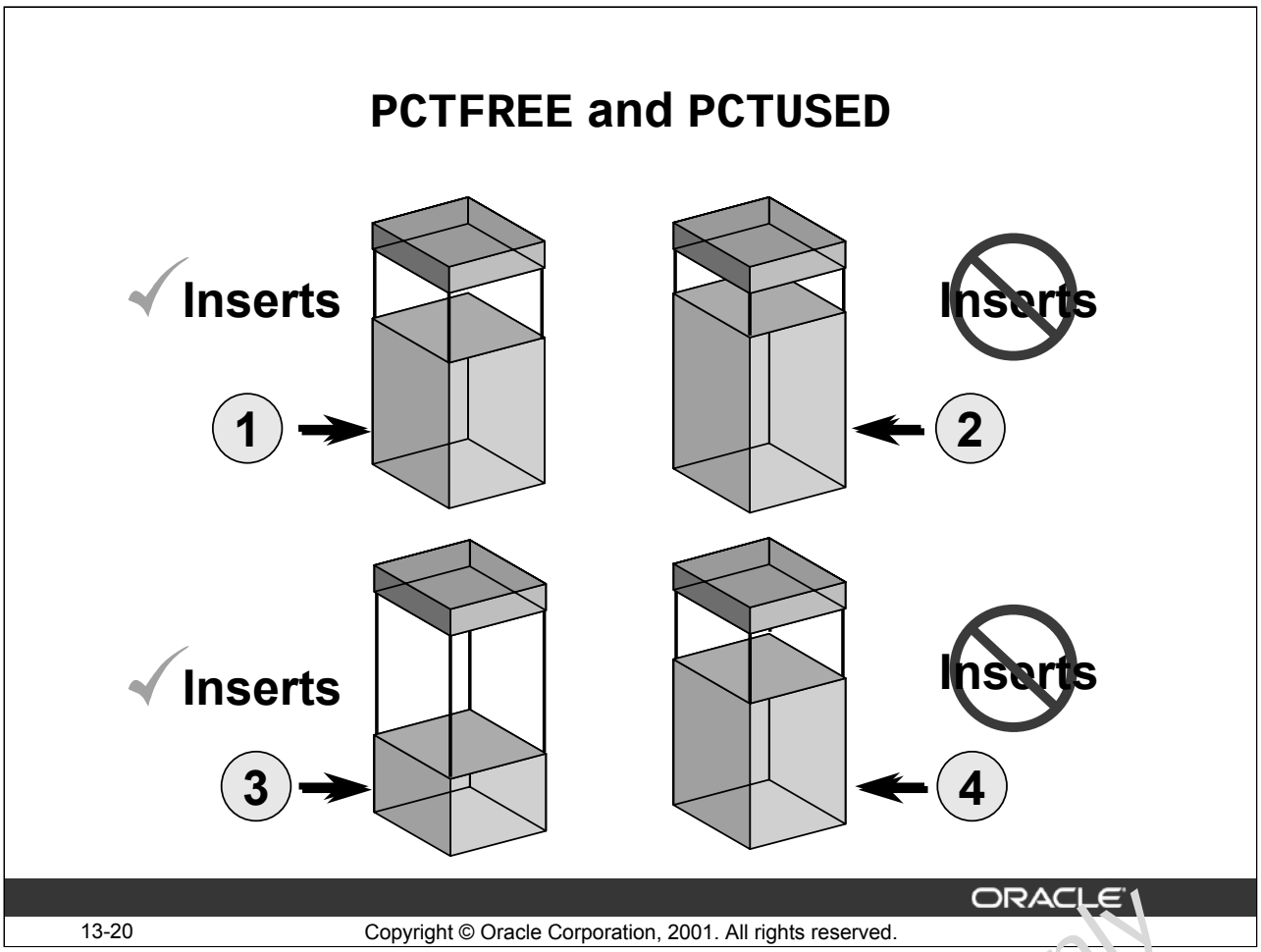
- There is less overhead and thus more room to store useful data.
- Large blocks are good for sequential reads.
- Large blocks are good for very large rows.
- Larger blocks improve the performance of index reads. The larger blocks can hold more index entries in each block, which reduces the number of 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.



How PCTFREE and PCTUSED Work Together

You use two space management parameters, PCTFREE and PCTUSED, 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 PCTFREE set to 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:

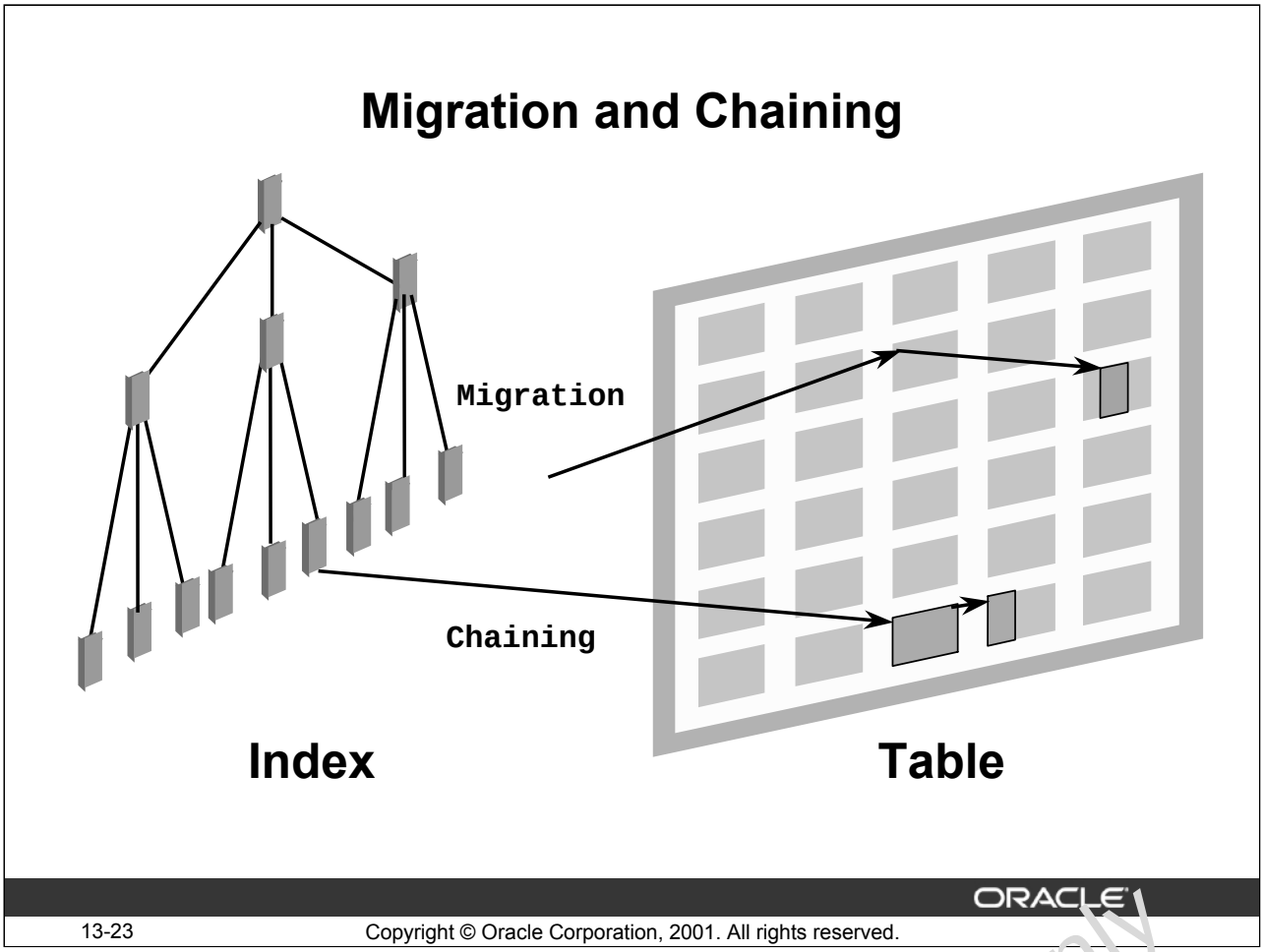
UPD = Average amount added by updates, in bytes. This is determined by subtracting the average row length of the insertcurrent average row length.
Average row length = Get this value from the AVG_ROW_LEN column of DBA_TABLES, which you retrieve after running ANALYZE
Rows = Number of rows to be deleted 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 can 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

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 in to 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 (continued)

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 by 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 by using 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:

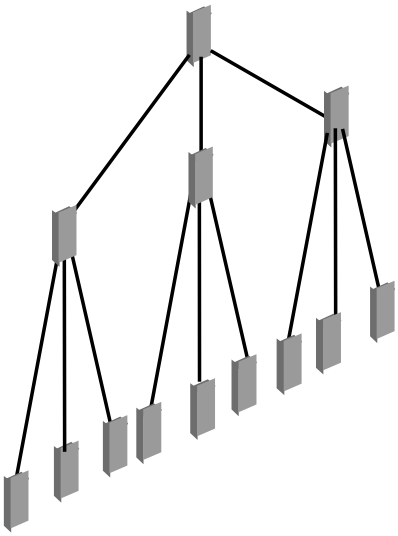
```
/* Get the name of the table with migrated rows */
accept table_name prompt 'Enter the name of the table with
  migrated rows: '
/* Clean up from last execution */
set echo off
drop table migrated_rows;
drop table chained_rows;

/* 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;
```

Eliminating Migrated Rows (continued)

```
/* 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.**

ORACLE

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;
```

ORACLE

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).

Rebuilding Indexes (continued)

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;
```

ORACLE

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**

ORACLE

Practice 13

Throughout this practice Enterprise Manager can be used if desired. SQL Worksheet can be used instead of SQL*Plus, and there are many uses for the Enterprise Manager Console. (Solutions for Enterprise Manager can be found in Appendix B).

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 SQL*Plus 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$enqueue_stat` for eq_type 'ST' in order to determine if the value of total_wait# for the ST enqueue, which is the space management enqueue.
   N**ote:** Record the difference in the number of waits for the ST enqueue for extent management using a dictionary managed tablespace. This value is found by subtracting the first wait value (from practice 13-04) from the second wait value (from practice 13-06).
7. Create a new locally managed tablespace `TEST`, name the data file `test01.dbf`, and place it in the directory `$HOME/ORADATA/u03`. Set the size to 120 MB, and a uniform extent size of 20 KB.
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$enqueue_stat` view.
10. Exit out of the SQL*Plus session and change directory to `$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.

Practice 13 (continued)

11. Again examine, and record the final `total_wait#` for 'ST' in the v$enqueue_stat view.
    Note: Record the difference in the `total_wait#` for the ST enqueue for extent management using a locally managed tablespace. This value is found by subtracting the first wait value (from practice 13-09) from the second wait value (from practice 13-11). Compare the two results for the different tablespaces. The locally managed tablespace would be far less contentious with extent management since it is managing the space within the tablespace itself.
12. Connect as user hr/hr and run the script `$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`.

14

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**
- **Partitioning methods**
- **The `DBMS_STATS` procedure**
- **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.**

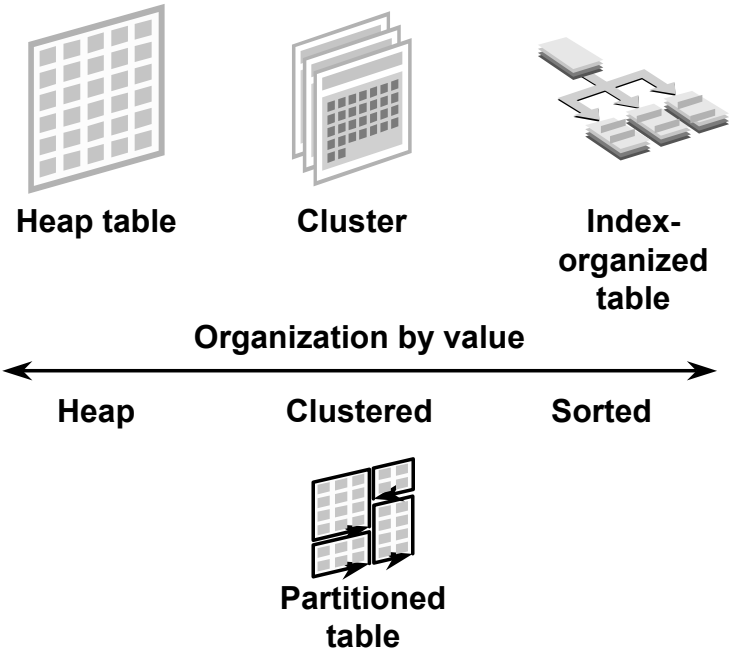
ORACLE

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 on 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

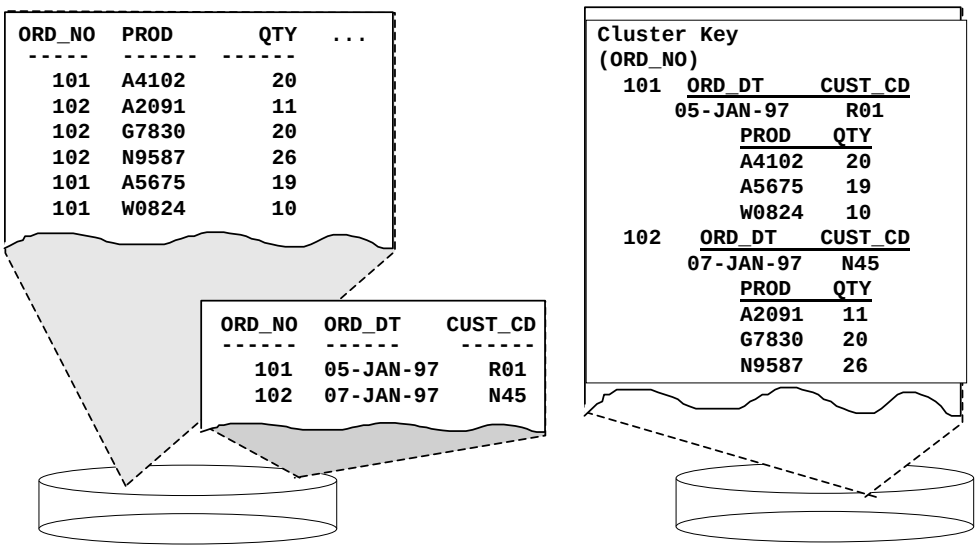
If large queries are occurring, you should investigate having multiple server processes distribute the workload between them. Using multiple server processes to distribute the table between disks and controllers will lessen contention and therefore enhance performance.

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



Unclustered `orders` and `order_item` tables

Clustered `orders` and `order_item` tables

ORACLE

Definition of Clusters

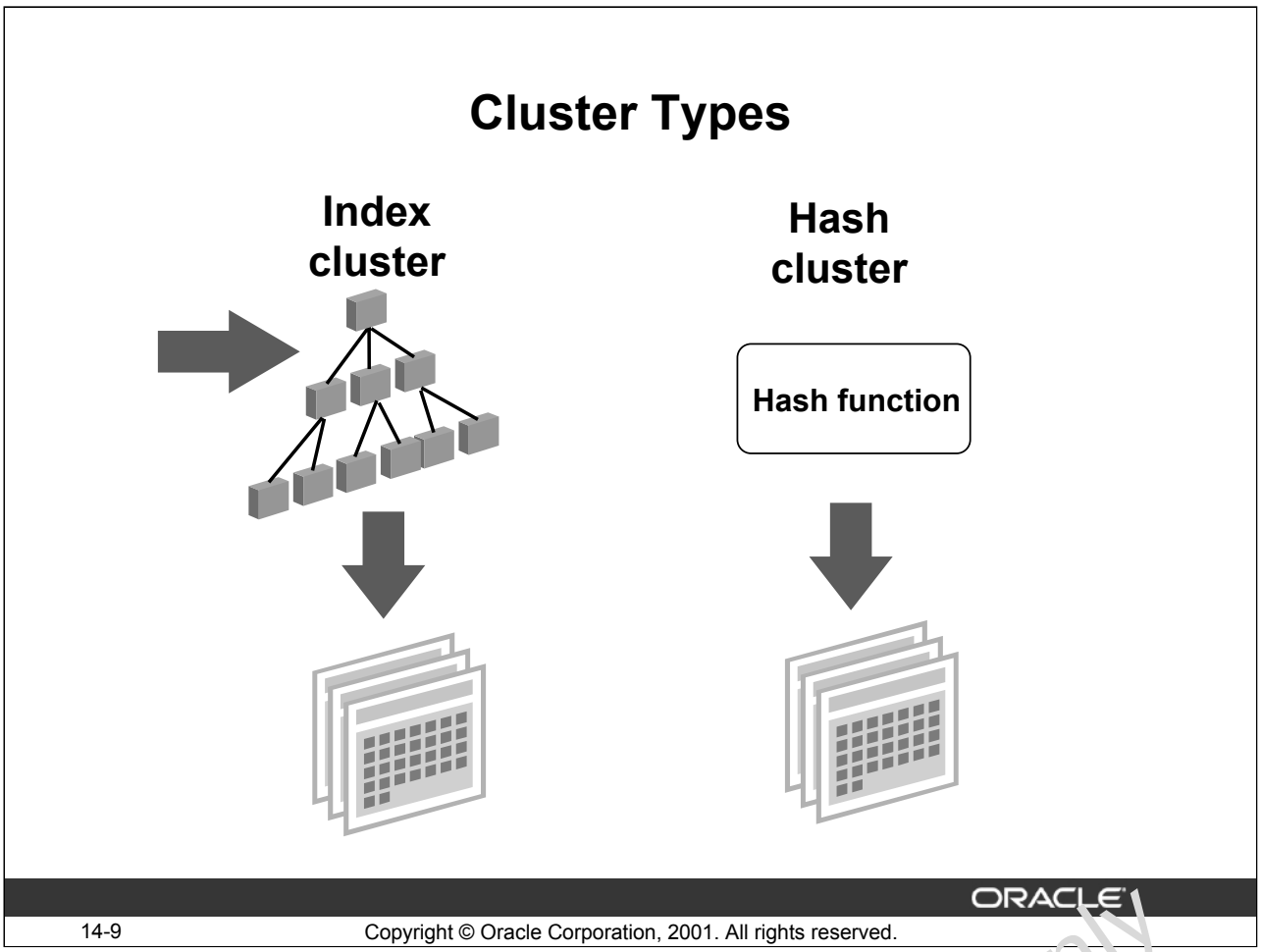
A cluster is a group of one or more tables that share the same data blocks because they share common columns and are often used together in join queries. Storing tables in clusters allows a DBA to denormalize data that is transparent to the end user and programmer.

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.



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

ORACLE

When Not to Use Clusters

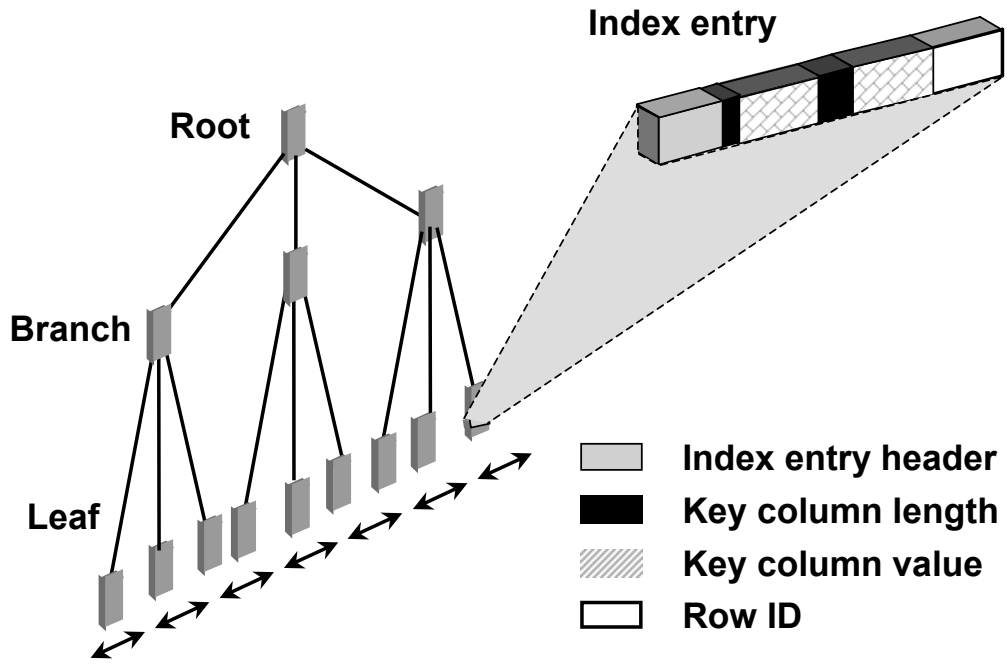
- If a full scan is executed often on one of the clustered tables: This table is stored on more blocks than if it had been created alone.
- If the data for all rows of a cluster key value exceeds one or two Oracle blocks: In order to access an individual row in a clustered key table, the Oracle server 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.

Hash and Index Clusters require a lot of planning before being used. There may be more performance overhead involved for major operations like bulk (direct path) inserts, and rebuilds.

B-Tree Indexes



ORACLE

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. If you select 5% of the data it may mean visiting just about every portion of the table anyway making it less efficient than a full scan. On the other hand, an index can be used to eliminate a potentially expensive sort when many rows are selected, or if 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.

How to Solve B-Tree Index Performance Degradation

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.

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.

ORACLE

Key Compression

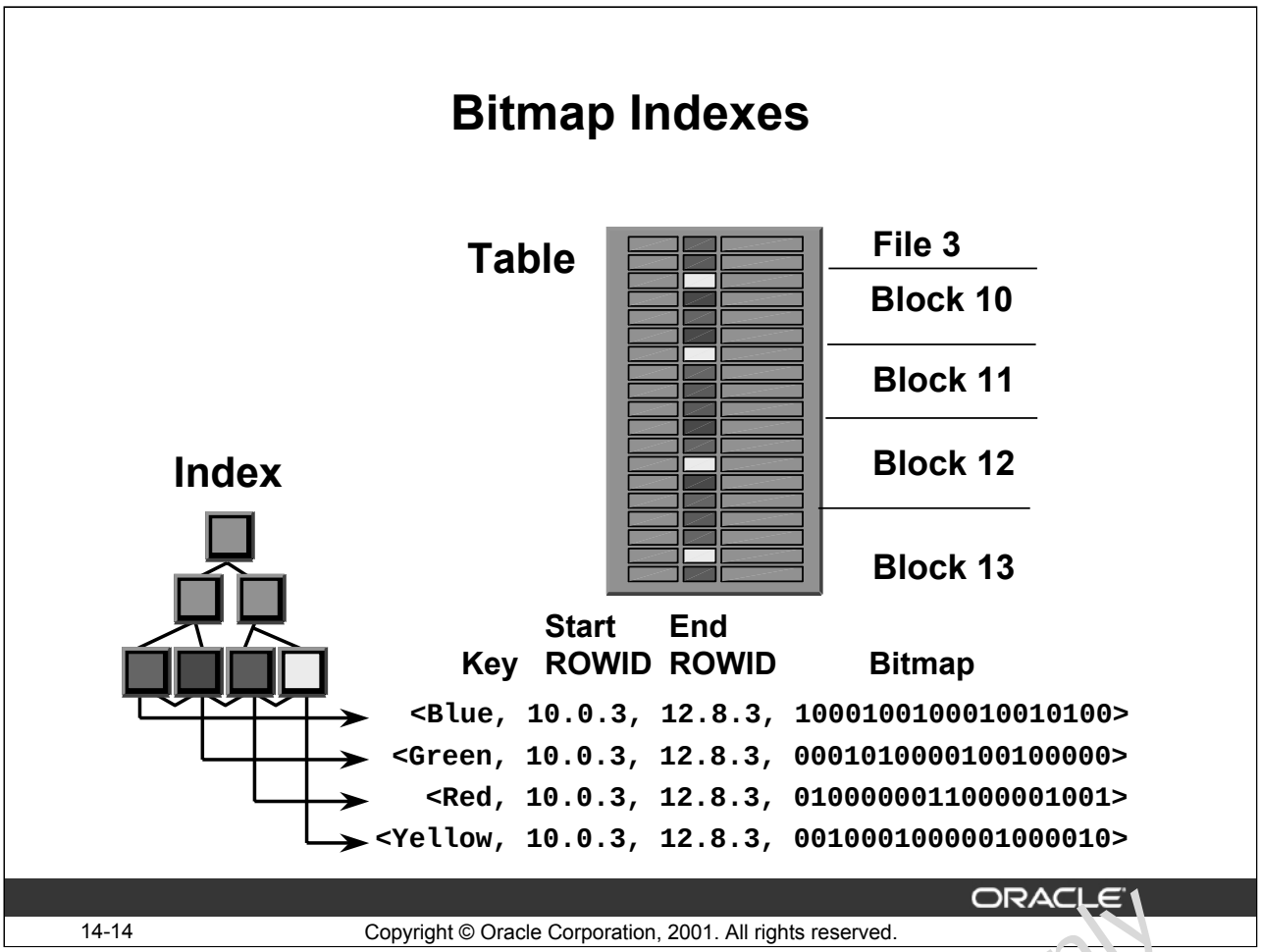
Specify COMPRESS to enable key compression, which eliminates repeated occurrence of key column values and may substantially reduce storage. Use integer to specify the prefix length (number of prefix columns to compress). For unique indexes, the valid range of prefix length values is from 1 to the number 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

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**

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

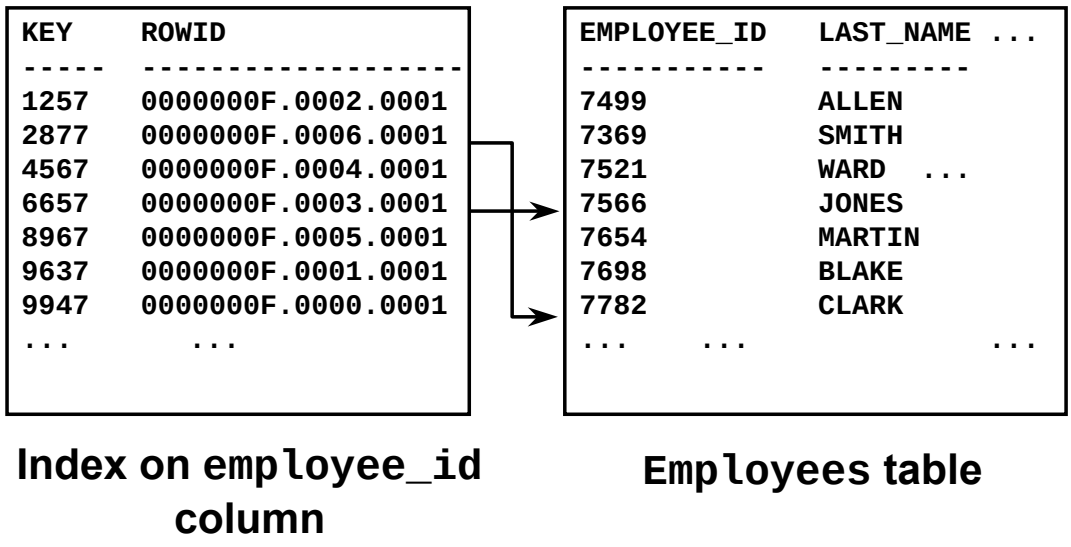
In a data warehousing environment, data is usually maintained by way of bulk inserts and updates. Index maintenance is deferred until the end of each DML operation. For example, if you insert 1,000 rows, then the inserted rows are placed into a sort buffer, and then the updates of all 1,000 index entries 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



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;
```

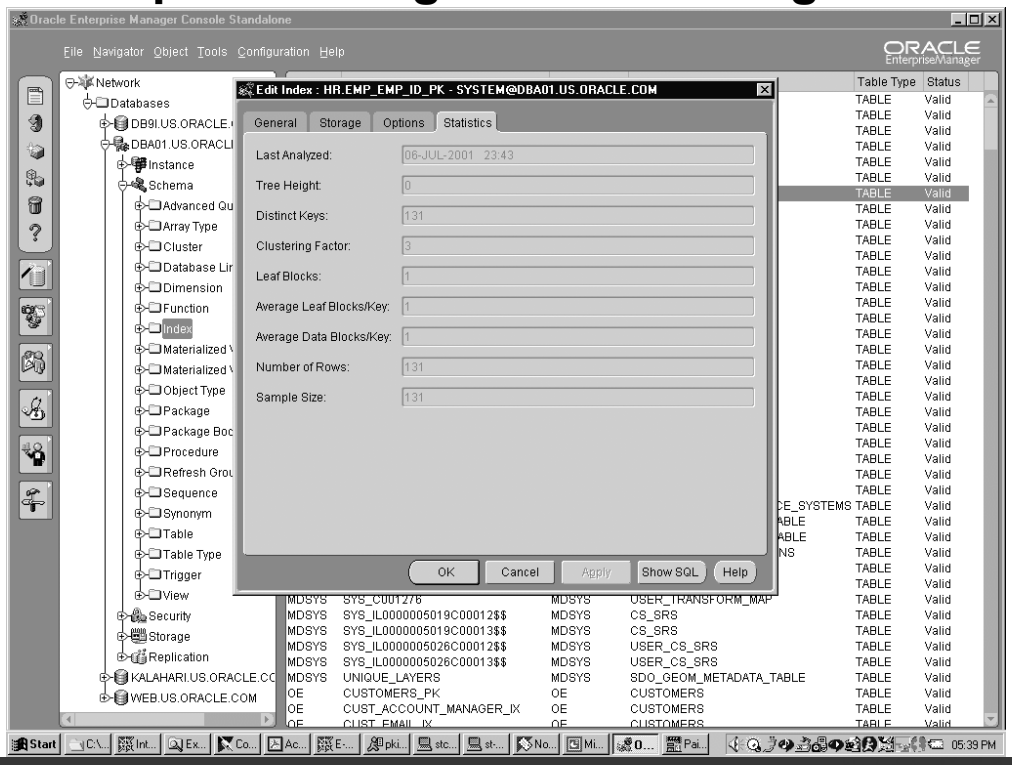
ORACLE

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
```

Enterprise Manager: Index Management

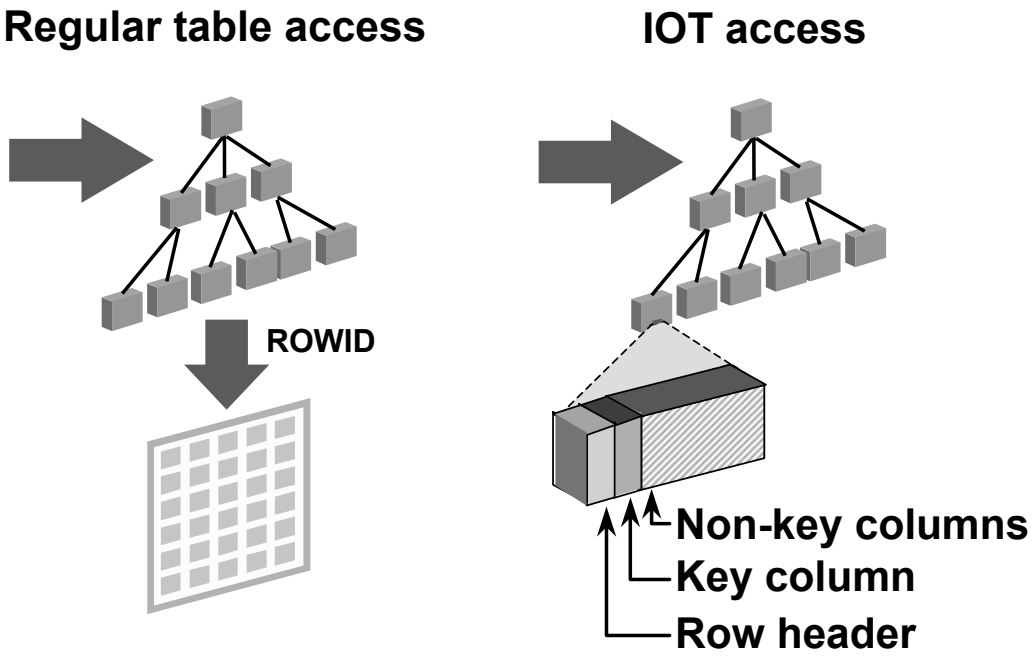


ORACLE

Enterprise Manager: Index Management

Use the Enterprise Manager Console in order to provide management for different objects stored within the database. In the example shown the statistics collected for an index are examined.

Index-Organized Tables



ORACLE

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**
- **IOTs have the following restrictions:**
  - **Must have a primary key**
  - **Cannot be clustered**

ORACLE

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;
```

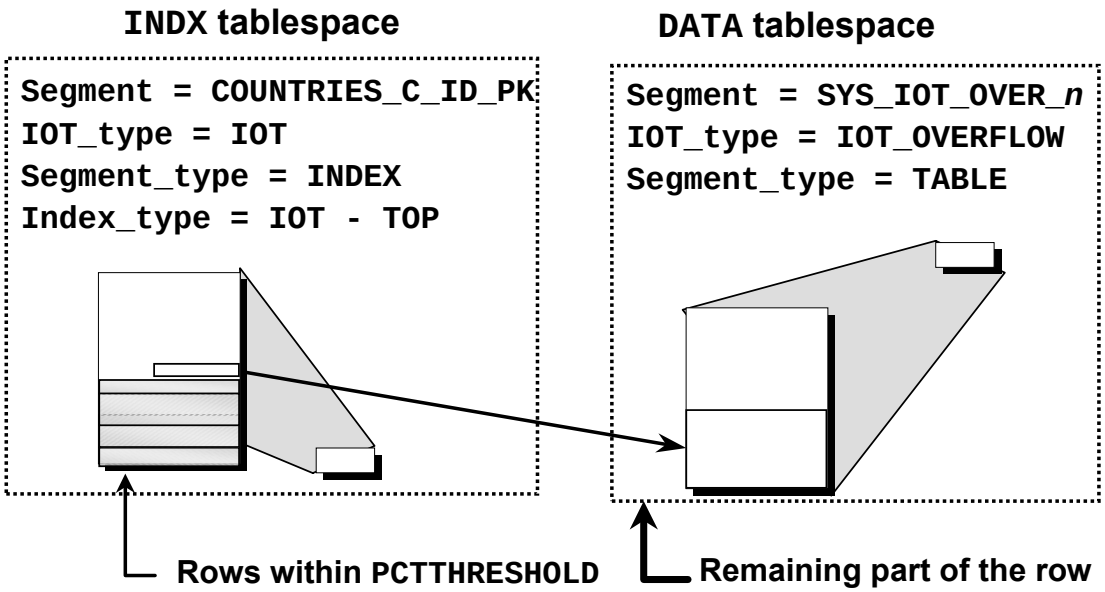

Creating Index-Organized Tables

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.

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
```

ORACLE

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;
```


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.**
- **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.**
- **Use the `MINIMIZE RECORDS_PER_BLOCK` clause of `ALTER TABLE` for the Mapping Table**

ORACLE

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 is analyzed.

To rebuild the mapping table, use the `ALTER TABLE` command specifying the `MAPPING TABLE UPDATE BLOCK REFERENCES` clause.

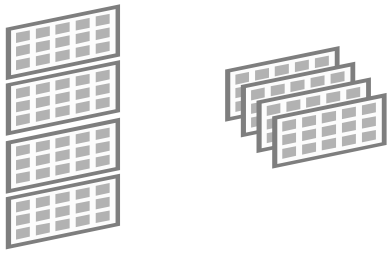
Oracle recommends using `MINIMIZE RECORDS_PER_BLOCK` on mapping tables that are heavily indexed with bitmap indexes. When enabled Oracle restricts the number of records in any block of the mapping table to the maximum number of records in any block of the indexed table. This enables the bitmap index to allocate fewer bits per block and results in smaller bitmap indexes.

The `NOMINIMIZE RECORDS_PER_BLOCK` clause disables this optimization.

Partitioning Methods

The following partitioning methods are available:

- **Range**
- **Hash**
- **Composite**
- **List**



Range partitioning | **Hash partitioning** | **Composite partitioning** | **List partitioning**

ORACLE

Partitioning Methods

A major driving force for supporting partitioned tables and indexes is the dramatic increase in the size of these database objects. This capability reduces down time (because of scheduled maintenance or data failures), improved performance through partition pruning, and improved manageability and ease of configuration.

Range Partitioning

Range partitioning uses ranges of column values to map rows to partitions. Partitioning by range is well suited for historical databases. However, it is not always possible to know beforehand how much data will map into a given range, and in some cases, sizes of partitions may differ quite substantially, resulting in suboptimal performance for certain operations like parallel DML.

Hash Partitioning

This method uses a hash function on the partitioning columns to stripe data into partitions. It controls the physical placement of data across a fixed number of partitions and gives you a highly tunable method of data placement.

Composite Partitioning

This method partitions data by using the range method and, within each partition, subpartitions it by using the hash method. This type of partitioning supports historical operations data at the partition level and parallelism (parallel DML) and data placement at the subpartition level.

List Partitioning

The LIST method allows explicit control over how rows map to partitions. This is done by specifying a list of discrete values for the partitioning column in the description for each partition.

LIST partitioning is different from RANGE partitioning where a range of values is associated with a partition, and from HASH partitioning where the user has no control of the row-to-partition mapping. This partition method allows the modeling of data-distributions that follow discrete values that are unordered and unrelated sets of data. These can be grouped and organized together very naturally, using LIST partitioning.

Range Partitioning Example

```
CREATE TABLE sales
  (acct_no      NUMBER(5),
   person       VARCHAR2(30),
   sales_amount NUMBER(8),
   week_no      NUMBER(2))                    (1)
   PARTITION BY RANGE (week_no)              (2)
    (PARTITION P1 VALUES LESS THAN (4)  TABLESPACE data0,   (3)
     PARTITION P2 VALUES LESS THAN (8)  TABLESPACE data1,
     ....
     PARTITION P13 VALUES LESS THAN (53) TABLESPACE data12
);
```

1. **The partition key is week_no**
2. **VALUES LESS THAN must be specified as a literal**
3. **Physical attributes can be set per partition**

Range Partitioning Example

Range partitioning maps rows to partitions based on ranges of column values. The partition key can be a composite key of up to 16 columns. In each partition, all rows have partition keys that compare less than the partition bound for that partition. You cannot have gaps in the ranges; for example, 1–3, 5–6, 8–10 is not allowed.

In the VALUES LESS THAN specification, you must either use a literal, the TO_DATE, or the RPAD function with constant arguments. If a table or index is partitioned on a column that has the DATE data type, you must specify the century with the year in the TO_DATE format mask or in the NLS format mask. The NLS date format is determined implicitly by NLS_TERRITORY or explicitly by NLS_DATE_FORMAT.

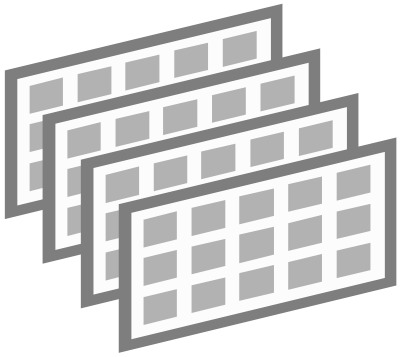
If you attempt to insert a row into a table such as SALES in the example, and the row's partitioning key binary value is greater than or equal to the highest partition bound in the table, the insert fails with the following error message:

```
ORA-14400: inserted partition key is beyond highest legal
partition key.
```

To avoid this, use MAXVALUE as the highest bound of the last partition to allow for all possible values to be inserted including NULL which is considered to sort greater than any other value except MAXVALUE.

Note: A partitioned table support any data types except LONG and LONG RAW. You can specify up to 64K-1 partitions.

Hash Partitioning Overview



- Easy to Implement
- Enables better performance for PDML and partition-wise join
- Inserts rows into partitions automatically based on the hash of the partition key
- Supports (hash) local indexes
- Does not support (hash) global indexes

ORACLE

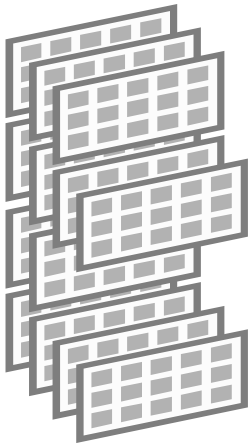
Hash Partitioning Concept

Although range partitioning is interesting for historical databases, it may not be the best choice for other purposes. Hash partitioning is a new partitioning method introduced in Oracle8*i* that uses a hash function on the partitioning columns to stripe data into partitions. Hash partitioning enables data to be easily partitioned for performance reasons such as parallel DML when sizes of range partitions would differ substantially.

Due to internal algorithms, the number of partitions should be a power of two (2, 4, 8, and so on) to obtain the most even data distribution.

In addition, cardinality is important for the partition key's choice. The partition key should be unique or near unique. Because a table may contain several near-unique keys, the other consideration for choosing the partition key is applications. Hash partitioning can improve the performance of single-key lookups, but it does not improve range lookups. More importantly, hash partitioning can enable partition-wise joins, and this should be the primary performance consideration when choosing a hash-partition key.

Composite Partitioned Table Overview



- **Ideal for both historical data and data placement**
- **Provides high availability and manageability, like range partitioning**
- **Improves performance for parallel DML, and supports partition-wise joins**
- **Allows more granular partition elimination**
- **Supports composite local indexes**
- **Does not support composite global indexes**

ORACLE

Composite Partitioned Tables

There is a need for partitioning methods which are both well-suited for historical data and simplify management for support of parallelism. One method is *composite* partitioning.

Under composite partitioning, an object consists of one or more partitions each of which in turn consists of one or more subpartitions.

The Oracle server supports composite range and hash partitioning. That is, the Oracle server partitions data by using the *range* method and, within each partition, subpartitions it by using the *hash* method.

Partitions (not subpartitions) using composite partitioning method have logical meaning defined by their partition bounds but no physical presence, because all data mapped into a partition resides in segments of subpartitions associated with it.

This type of partitioning supports historical operations data at the partition level and parallelism (PDML) and data placement at the subpartition level.

List Partitioning Example

```
CREATE TABLE locations
(location_id, street_address, postal_code, city,
state_province, country_id)
PARTITION BY LIST (state_province)
STORAGE(INITIAL 10K NEXT 20K) TABLESPACE tbs5
(PARTITION region_east
 VALUES('MA','NY','CT','ME','MD','VA','PA','NJ')
 STORAGE (INITIAL 20K NEXT 40K PCTINCREASE 50)
 TABLESPACE tbs8,
PARTITION region_west
 VALUES('CA','AZ','NM','OR','WA','UT','NV','CO')
 PCTFREE 25 NOLOGGING,
PARTITION region_south
 VALUES('TX','KY','TN','LA','MS','AR','AL','GA'),
PARTITION region_central
 VALUES('OH','ND','SD','MO','IL','MI',NULL,'IA')
);
```

ORACLE

List Partitioning Example

The details of list partitioning can best be described with an example. In this case you want to partition the `LOCATIONS` table by region; that is, you want to group states together according to their geographical location.

A row is mapped to a partition by checking whether the value of the partitioning column for a row falls within the set of values that describes the partition.

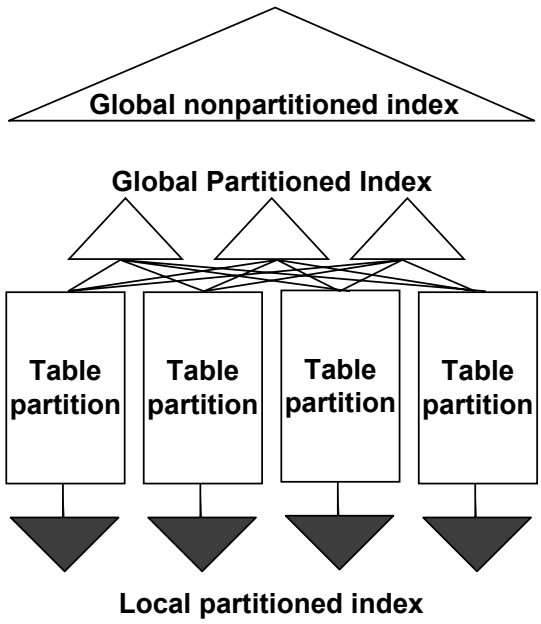
For example, the following rows will be inserted as follows

- (1500,'2011 Interiors Blvd', '99236','South San Francisco', 'CA', 'US') maps to partition region_west
- (5000,'Chemin de la Fanee','13340','ROGNES','BR','FR') will not map to any partitions in the table and thus will fail with an `ORA-14400` error. This partitioning method is useful when the partitioning column consists of bounded set of discrete values. However, in this release, it is not possible to define a catch-all partition (like `MAXVALUE` for range partitioning). This will be available in a later release of Oracle9*i*.

In the above example, the partitions with specified physical attributes will override the table-level defaults. However, any partition with unspecified attributes inherits their physical attributes from the table-level defaults.

Note: For formatting reasons, the column data types are not specified. Refer to the Oracle9*i* Sample Schema definition for more details on those data types.

Partitioned Indexes for Scalable Access



- Indexes can be partitioned delivering improvements in:
  - Manageability
  - Availability
  - Performance and scalability
- Enables parallel index scans
- Choices in index configuration:
  - Local prefixed index
  - Local nonprefixed index
  - Global prefixed index
  - Global nonpartitioned index
- Flexibility to suit a variety of access patterns, index sizes

ORACLE

Prefixed Partitioned Indexes

An index is prefixed if it is partitioned on a left prefix of the index columns.

Prefixed indexes can be unique or nonunique.

Non-prefixed Partitioned Indexes

An index is nonprefixed if it is not partitioned on a left prefix of the index columns.

Nonprefixed indexes can be unique only if their partitioning key is a subset of the index key.

Local Partitioned Indexes

In a local index, all keys in a particular index fragment refer only to rows stored in a single underlying table fragment. A local index is created by specifying the LOCAL attribute. The Oracle server constructs the local index so that it is equipartitioned with the underlying table. To ensure that the index remains equipartitioned with the table, the Oracle server also maintains the index partitioning automatically when partitions in the underlying table are added, dropped, merged, or split, or when hash partitions or subpartitions are added or coalesced.

A bitmap index on a partitioned table must be a local index.

Global Partitioned Indexes

In a global partitioned index, the keys in a particular index partition may refer to rows stored in more than one underlying table partition or subpartition. A global index can only be range-partitioned, but it can be defined on any type of partitioned table. A global index is created by specifying the GLOBAL attribute. The highest partition bound of a global index must be MAXVALUE. This insures that all rows in the underlying table can be represented in the index.

The database administrator is responsible for defining the initial partitioning of a global index at creation and for maintaining the partitioning over time. Index partitions can be merged or split as necessary.

A global index can be equipartitioned with the underlying table, but the Oracle server does not take advantage of the equipartitioning when generating query plans or executing partition maintenance operations. An index that is equipartitioned with the underlying table should be created as LOCAL.

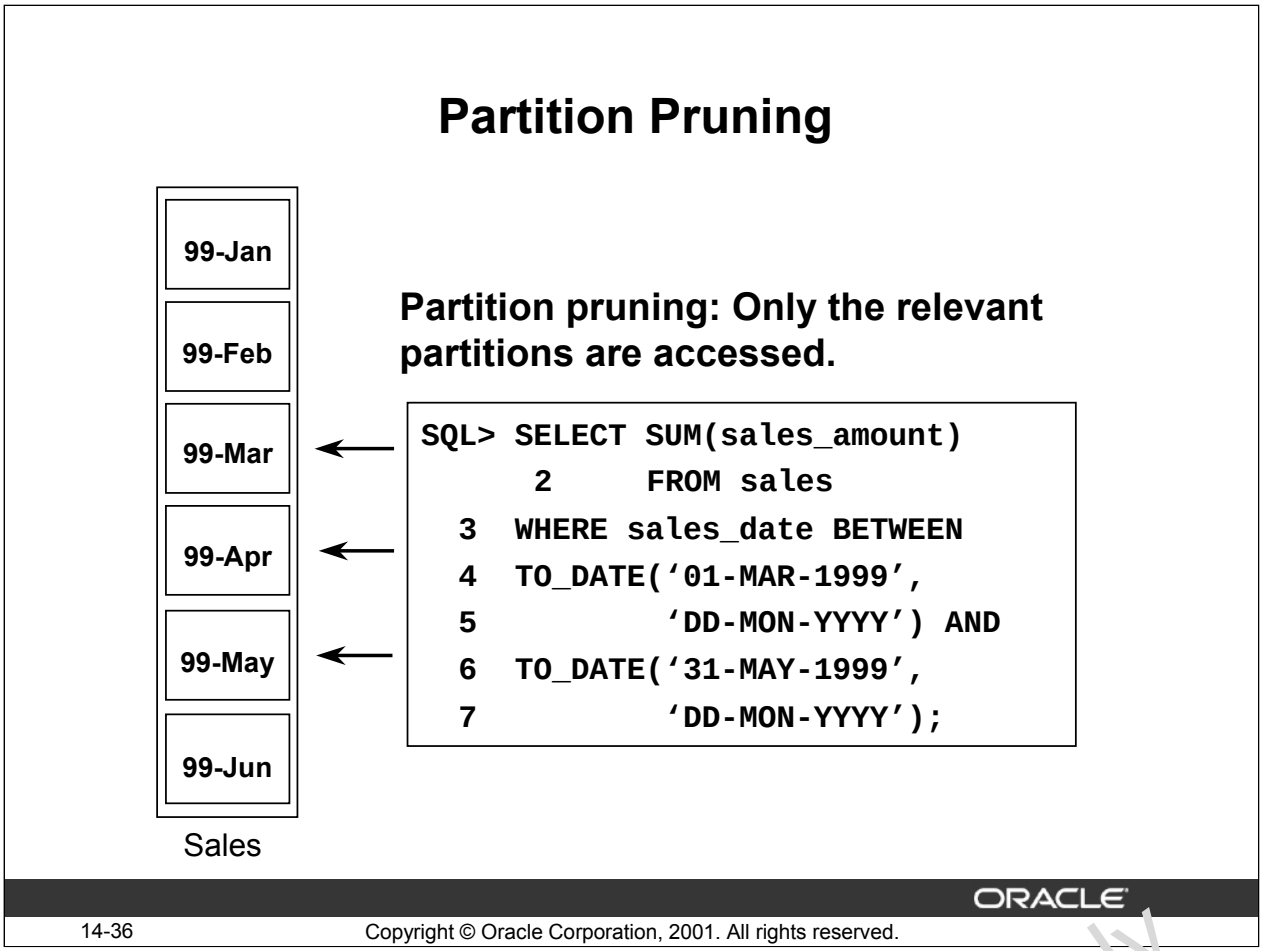
Nonprefixed global indexes are not supported because they have no application in reality.

Nonpartitioned Indexes

Nonpartitioned indexes are treated as global prefixed non-partitioned indexes.

Guidelines for Partitioned Indexes

When deciding how to partition indexes on a table, consider the mix of applications that need to access the table. There is a trade-off between performance on the one hand and availability and manageability on the other. You should create local indexes on partitioned tables unless there is a justified performance requirement for a global index. This is because most of the partitioned tables maintenance operations invalidate all the global index partitions implying their complete rebuild.



Partition Pruning

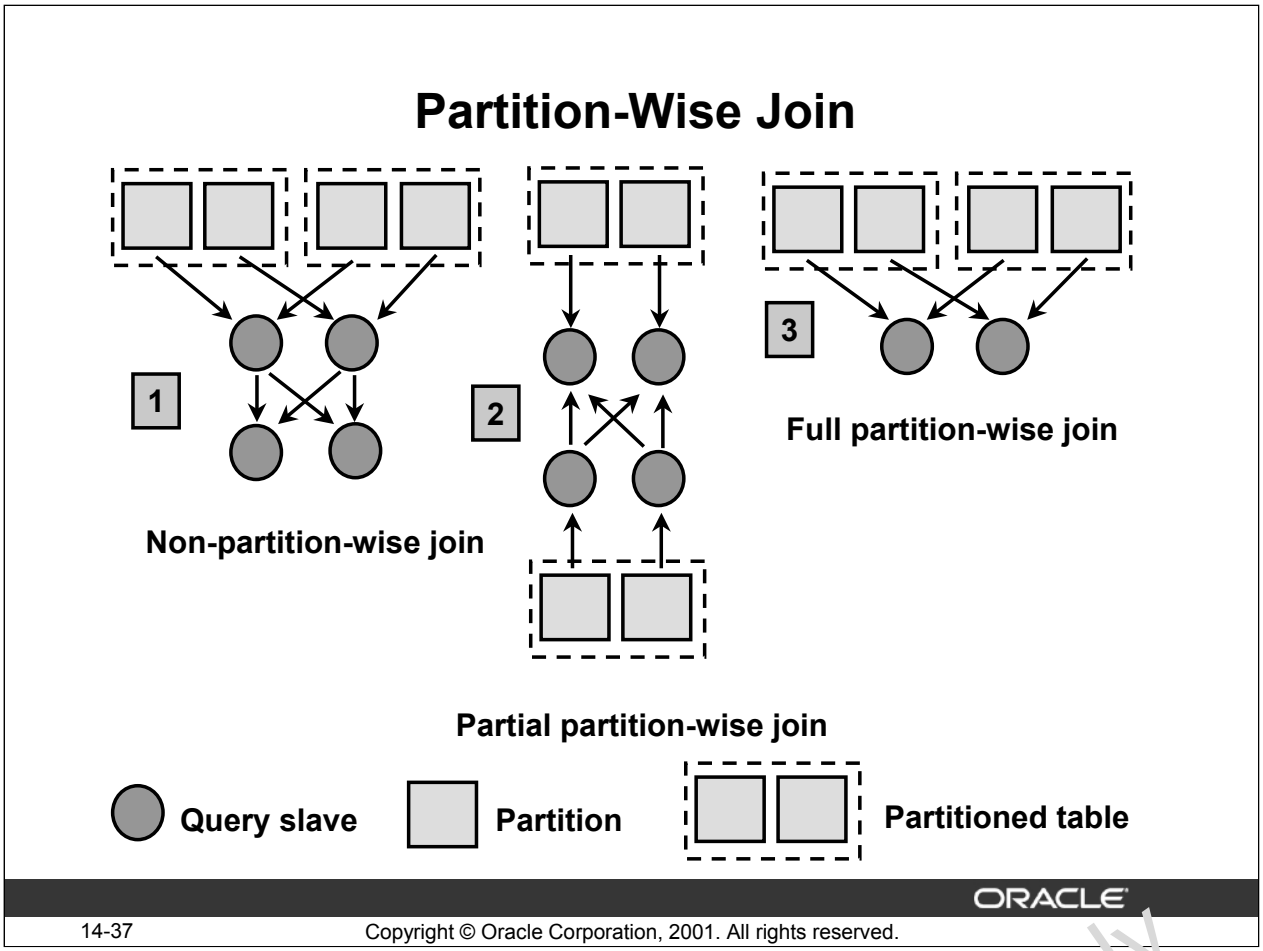
Depending on the SQL statement, the Oracle server can explicitly recognize partitions and subpartitions (of tables and indexes) that need to be accessed and the ones that can be eliminated. This optimization is called partition *pruning*. This can result in substantial improvements in query performance. However, the optimizer cannot prune partitions if the SQL statement applies a function to the partitioning column.

If you partition the index and table on different columns (with a global, partitioned index), partition pruning also eliminates index partitions even when the underlying table's partitions cannot be eliminated. However, the optimizer cannot use an index if the SQL statement applies a function to the indexed column, unless it is a function-based index.

The Oracle server considers only the relevant partitions for such predicates as as `c in (10,30)` or `c=10 or c=30`

For hash partitioning, partition pruning is limited to using equality or `IN-list` predicates.

The optimizer can also performs partition pruning when subquery predicates are used.



Partition-Wise Joins

Partition-wise joins minimize the amount of data exchanged among parallel slaves during parallel joins execution by taking into account data distribution. This significantly reduces response time and resource utilization, both in terms of CPU and memory. In Oracle Parallel Server environments, it can limit the data traffic over the interconnect (if the relevant partitions are "co-located"), which is vital for massive join operations.

Full Partition-Wise Join

A full partition-wise join is a join that can be performed on two tables which are equipartitioned in advance on their join keys. The Oracle server performs the joins sequentially if the query is executed serially, or if each pair is joined by a separate query slave if it is executed in parallel. The Oracle server can join the results of each parallel scan without the need to redistribute data. Partition-wise joins are limited by the number of partitions.

Partial Partition-Wise Join

A partial partition-wise join is a join that can be performed on two tables when only one of the tables is partitioned on their join keys. Other must be dynamically repartitioned to meet the partitioning scheme of the first one. It is only implemented when joins are run in parallel.

Note: The Oracle server supports partition-wise joins on range, hash, composite, or list partitioned tables.

Example

Consider the following typical query: “Find the records of all customers who bought more than 100 articles in quarter 3 of 1994.”

```
SELECT c_customer_name, COUNT(*)
FROM sales, customer
WHERE s_customerid = c_customerid
AND s_saledate BETWEEN TO_DATE(01-jul-1994, DD-MON-YYYY)
AND TO_DATE(01-oct-1994, DD-MON-YYYY)
GROUP BY c_customer_name HAVING
COUNT(*) > 100;
```

Suppose that the optimizer uses a hash join in this case. You can reduce the processing time for this join if both tables are partitioned correctly because this enables a partition-wise join. Multiple partitioning methods can be used, but not all of them activate the partition-wise join:

- Hash-Hash: `Customer` and `Sales` tables are equipartitioned by hash on `s_customerid` and `c_customerid` respectively. In this case, the technique used by the Oracle server to make this join parallel is number 3 in the partition-wise join slide. This is the most efficient. It can become even better in a cluster or MPP environment if each corresponding hash-partition are colocated (placed on the same local disk).
- Composite-Hash: The `sales` table is partitioned by range on `s_saledate` (each partition representing a quarter) and subpartitioned by hash on `s_customerid`. The customer table is partitioned by hash on `c_customerid` so that both tables are equipartitioned on the hash dimension. In this case, because the pruning on the `sales` table restricts the scan to the subpartitions corresponding to Q3 of 1994, the same technique as in the previous example can be used.
- Range-Hash: However, if the `sales` table is partitioned by range on `s_saledate` and the `customer` table is partitioned by hash on `c_customerid`, then the technique used by the Oracle server in this case is number 2 in the partition-wise join slide. Only a partial partition-wise join can be used.
- Range-Range: If both tables are range partitioned, then the only available technique is number 1 in the partition-wise join slide. This is probably the worst case.

Statistics Collection for Partitioned Objects

- You can gather object-, partition-, or subpartition level statistics.
- There are GLOBAL, or NON-GLOBAL, statistics.
- DBMS_STATS can gather global statistics at any level for tables only.
- It is not possible to gather:
  - Global histograms
  - Global statistics for indexes

ORACLE

Cost-Based Optimization and Partitioned Objects Statistics

Statistics can be gathered by partition or subpartition by using the DBMS_STATS package. The cost-based optimizer is always used for SQL statement accessing partitioned tables or indexes.

For partitioned objects, the Oracle server maintains separate sets of statistics: at the object level, the partition level, and the subpartition level. If a SQL statement accesses only one fragment, then the Oracle server uses the fragment level's statistics. If a SQL statement accesses multiple fragments, the Oracle server uses a single access path for all of these fragments and uses the statistics from the next higher level. (Here, a fragment can be a subpartition or a partition of a nonconposite object; subpartition is considered the lowest level, partition is the next level, and object is the last.)

DBMS_STATS always collects global statistics (except histograms and indexes) at any level and never merges them. Global statistics are obtained by considering multiple fragments as only one.

DBMS_STATS Examples

```
CALL DBMS_STATS.GATHER_TABLE_STATS(
           ownname => 'o901', tabname => 'sales',
           partname => 'feb99', granularity => 'partition');
```

```
CALL DBMS_STATS.GATHER_INDEX_STATS(
           ownname => 'o901', indname => 'isales',
           partname => 's1');
```

ORACLE

DBMS_STATS.GATHER_TABLE_STATS Procedure

This is a list of some important arguments of this procedure. Refer to the *Oracle9i Supplied PL/SQL Packages Reference* for a complete description:

- `ownname`: Schema of table to analyze
- `tabname`: Name of table
- `partname`: Name of partition or subpartition depending on granularity
- `method_opt`: Equivalent to the for-clause of the ANALYZE command for histograms
- `degree`: Degree of parallelism (`NULL` means use table default value.)
- `granularity`: Granularity of statistics to collect (only pertinent if the table is partitioned)
  - `DEFAULT`: Gather table- and partition-level statistics
  - `SUBPARTITION`: Gather subpartition-level statistics
  - `PARTITION`: Gather partition-level statistics
  - `GLOBAL`: Gather object-level statistics
  - `ALL`: Gather all (subpartition-, partition-, and object-level) statistics
- `cascade`: Gather statistics on the indexes for this table. Index statistics gathering is not made parallel.

Note: Except for indexes and histograms, gathered statistics are always global.

DBMS_STATS.GATHER_INDEX_STATS Procedure

This procedure gathers index statistics. It is equivalent to running ANALYZE INDEX. It does not execute in parallel:

- ownname: Schema of index to analyze
- indname: Name of index
- partname: Name of partition or subpartition
- estimate_percent: Percentage of rows to estimate (NULL means compute.)
- stattab: User stat table identifier describing where to save the original statistics
- statid: Identifier (optional) to associate with these statistics within stattab
- statown: Schema containing stattab (if different than ownname)

Note: The last four parameters are also available with the GATHER_TABLE_STATS procedure.

There are also two other procedure that can be used:

- GATHER_SCHEMA_STATS: This procedure gathers statistics for all objects in a schema.
- GATHER_DATABASE_STATS: This procedure gathers statistics for all objects in the database.

ANALYZE Statement

The ANALYZE statement can be used to:

- **VALIDATE STRUCTURE**
- **LIST CHAINED ROWS**
- **Collect statistics not used by the optimizer, such as information on free list blocks**
- **Sample a number (instead of a percentage) of rows**

```
ANALYZE TABLE hr.employees validate structure;
```

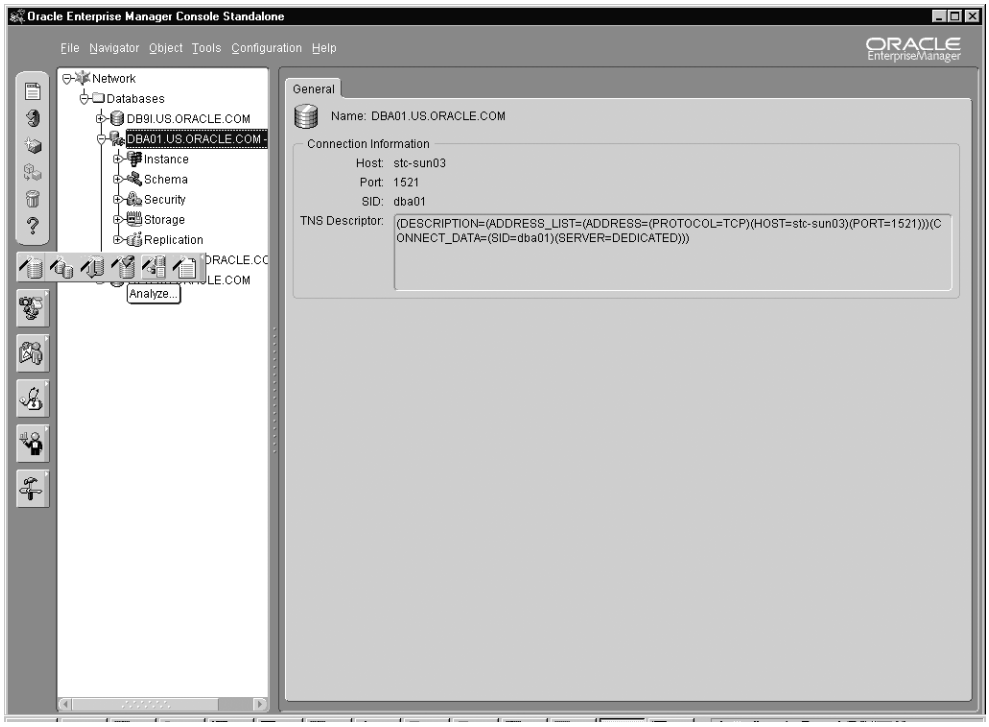
ORACLE

ANALYZE Statement

The ANALYZE statement is the predecessor of the DBMS_STATS package and can be used to collect similar information. The DBMS_STATS package is superior and is thus the recommended method of collecting statistics. However, the following cannot be collected using DBMS_STATS :

- To use the VALIDATE or LIST CHAINED ROWS clauses
- To sample a number (rather than a percentage) of rows
- To collect statistics not used by the optimizer (such as information on free list blocks)

Enterprise Manager: Collect Statistics



ORACLE

Enterprise Manager: Collect Statistics

By selecting Analyze in the Enterprise Manager Console will give the DBA the opportunity to collect statistics on various objects. There is also the option to use the DBMS_STATS package (recommended) or ANALYZE.

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)**

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 reexecuted 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;
```


Creating Materialized Views

Materialized views are an advantage over regular views in that the result of the view is stored for future use. This means that the query does have to be rebuilt every time the query is run.

One of the problems with materialized views is that the data stored in the view can become outdated with the next modification to the underlying tables. Therefore, it is required to refresh the data periodically.

Refreshing Materialized Views

The required parameters are:

- **A comma-delimited list of materialized views to refresh**
- **The refresh method: F-Fast, ?-Force, C-Complete**
- **Refresh after errors**
  - **TRUE : allows the process to continue after an error**
  - **FALSE: refresh will stop with errors. This is the default**
- **For warehouse refresh, set them to FALSE, 0,0,0.**
- **Atomic refresh**
  - **TRUE : all refreshes are done in one transaction.**
  - **FALSE : each refresh is a separate transaction.**

```
SQL> exec DBMS_MVIEW.REFRESH('SALES_MV', 'F',
          '', TRUE, FALSE, 0,0,0, FALSE);
```

ORACLE

Refreshing Materialized Views

Materialized views use the same internal mechanism as snapshots, and they 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 reexecuting 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.**

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 (continued)

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
```

ORACLE

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**
- **Hints: REWRITE and NOREWRITE**
- **Dimensions**

ORACLE

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
. . .
```

ORACLE

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.

DBMS_MVIEW Package

The package contains the following procedures:

- **EXPLAIN_MVIEW**
- **EXPLAIN_REWRITE**
- **REFRESH**
- **REFRESH_ALL_MVIEWS**

DBMS_MVIEW Package

EXPLAIN_MVIEW 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

You use this procedure 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.

REFRESH Procedure

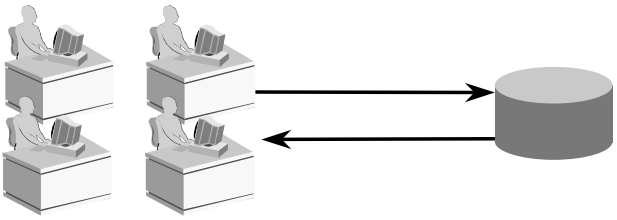
This procedure consistently refreshes one or more materialized views that are not members of the same refresh group.

REFRESH_ALL_MVIEWS Procedure

Refreshes all materialized views that do not reflect changes to their master table or master materialized view.

OLTP Systems

- **High-throughput, insert- and update-intensive**
- **Large, continuously growing data volume**
- **Concurrent access by many users**
- **Tuning goals:**
  - **Availability**
  - **Speed**
  - **Concurrency**
  - **Recoverability**



ORACLE

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**

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.

Materialized Views

- Materialized Views can reduce the amount of table joins on a system. There are not many opportunities for using Materialized Views in an OLTP system, most benefits are found in DSS.

OLTP Requirements (continued)

Index Organized Tables

- An IOT provides fast access to data, since the table data is stored within the index structure.

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.

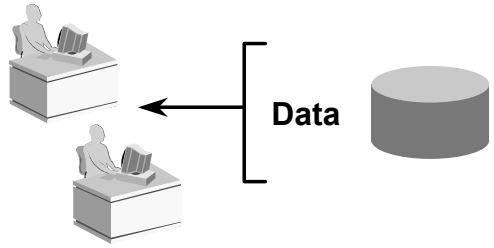
The `CURSOR_SHARING` Parameter

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` Oracle9*i* 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
  - Focus on SQL statement tuning



- The Parallel Query feature is designed for data warehouse environments.

ORACLE

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 this 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 timeframe.
- Focus on SQL statement tuning.

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 `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 is 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.**

ORACLE

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 the predicate of a query uses columns that have histograms generated. This means that if you have generated histogram statistics it is important for them to be accurate; otherwise the optimized execution path might not be chosen.

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 14

In this practice you will make use of the AUTOTRACE feature, and create the table `plan_table`. These are covered in detail in the lesson titled "SQL Statement Tuning." Through out this practice Enterprise Manager can be used if desired. SQL Worksheet can be used instead of SQL*Plus, and there are many uses for the Enterprise Manager Console. (Solutions for Enterprise Manager can be found in Appendix B).

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 14 (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/lab14_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.

15

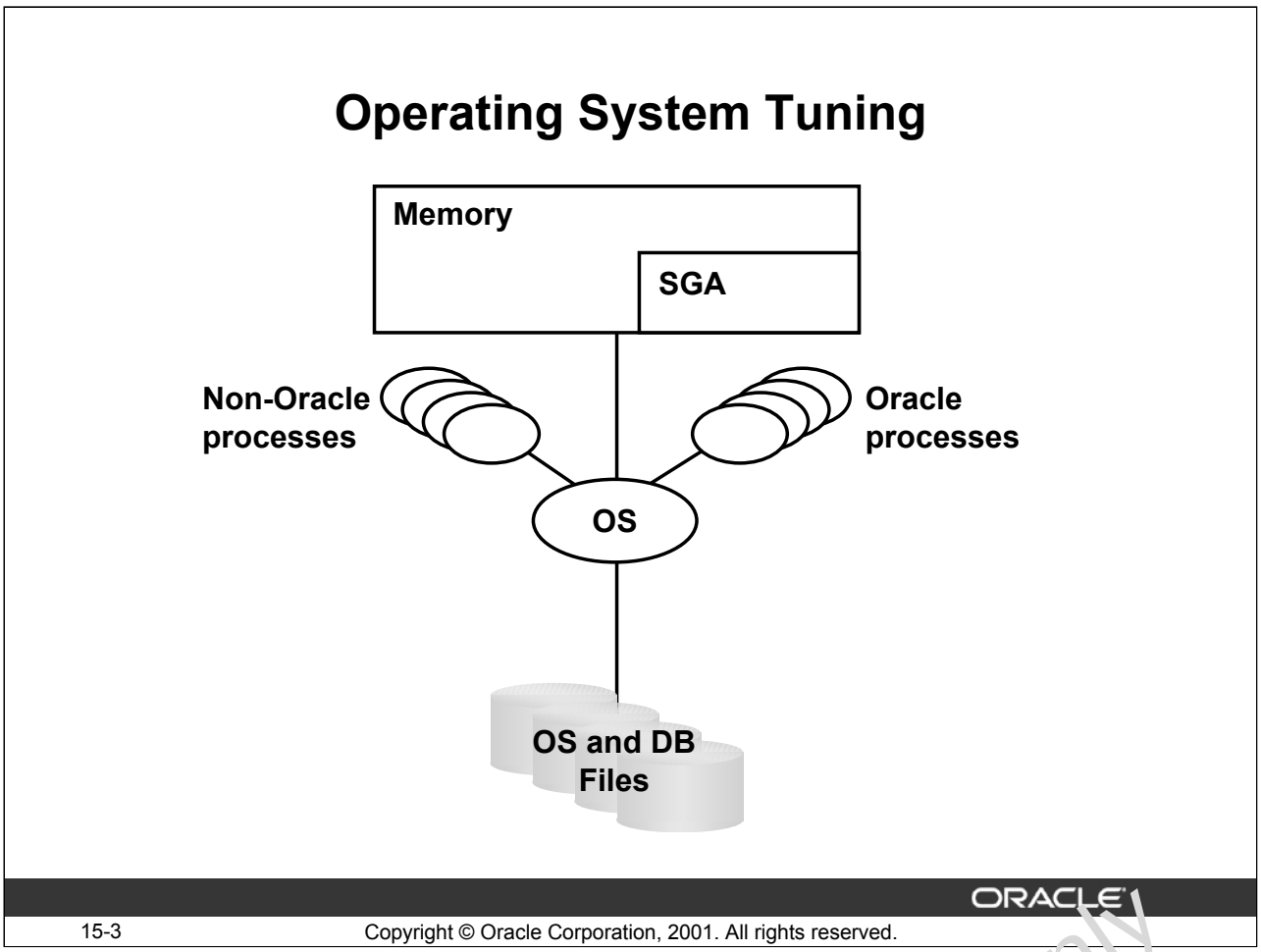
Tuning the Operating System and Using Resource Manager

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**

ORACLE



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:

- **Uni Processor 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

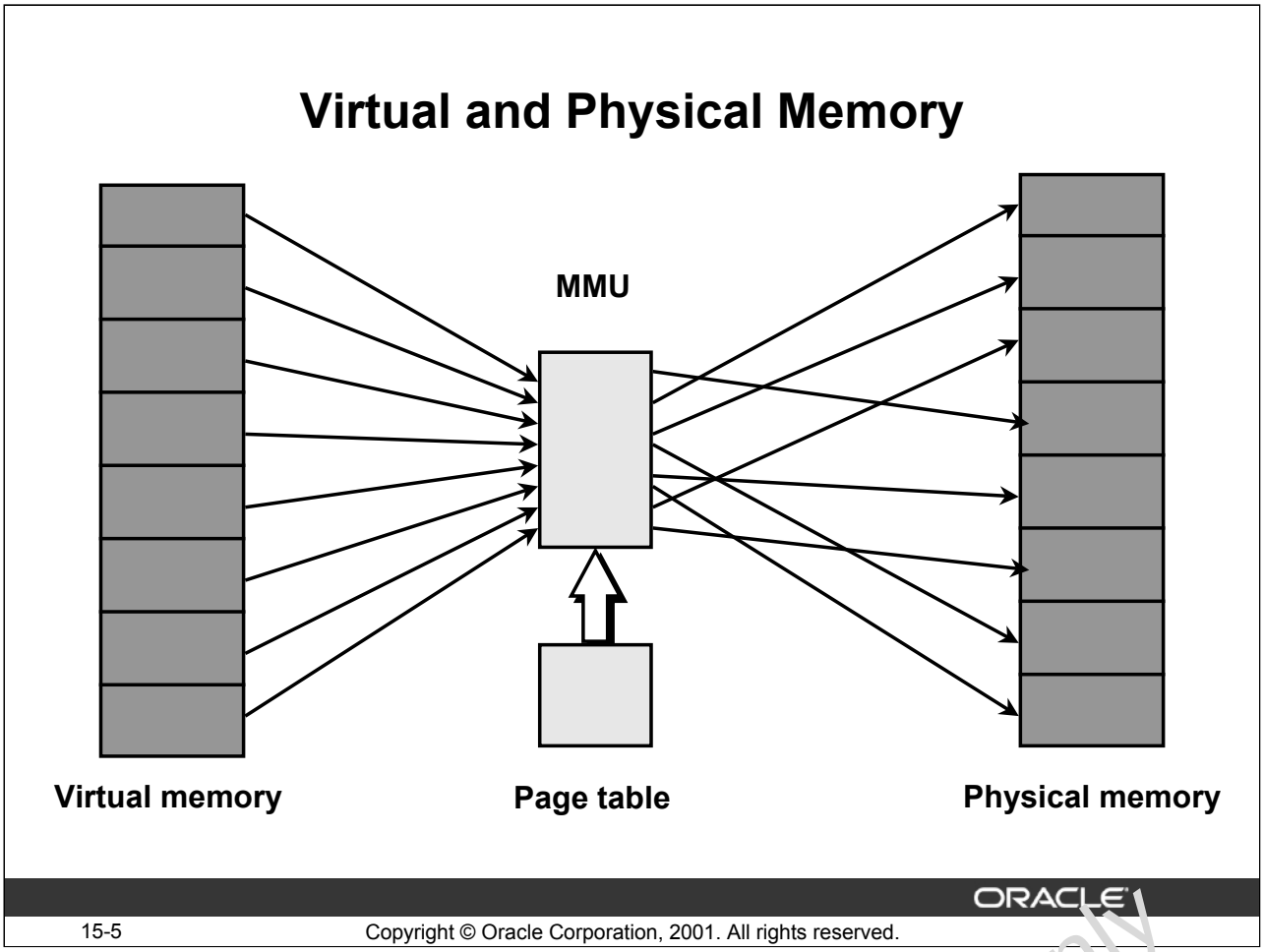
MPP systems consist of several nodes connected together. Each node has its own CPU, memory, bus, disks, and I/O system. Each node runs its own copy of the operating system. The number of nodes in an MPP system can range from two to even several thousands.

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.

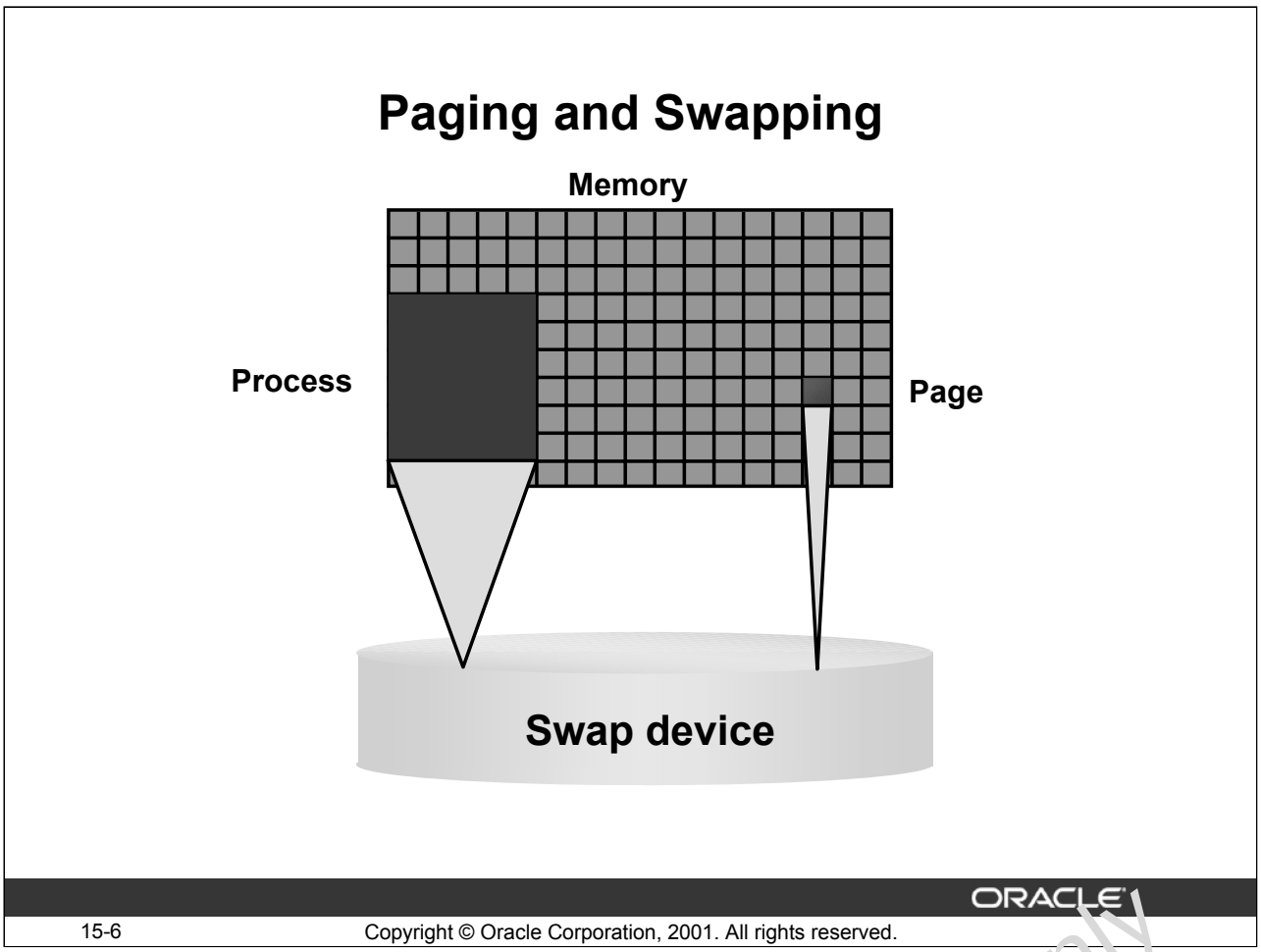
Clustered (Cluster) Systems

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 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

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 but only paging, and most others are swapping only as a last resort when the amount of free memory is 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

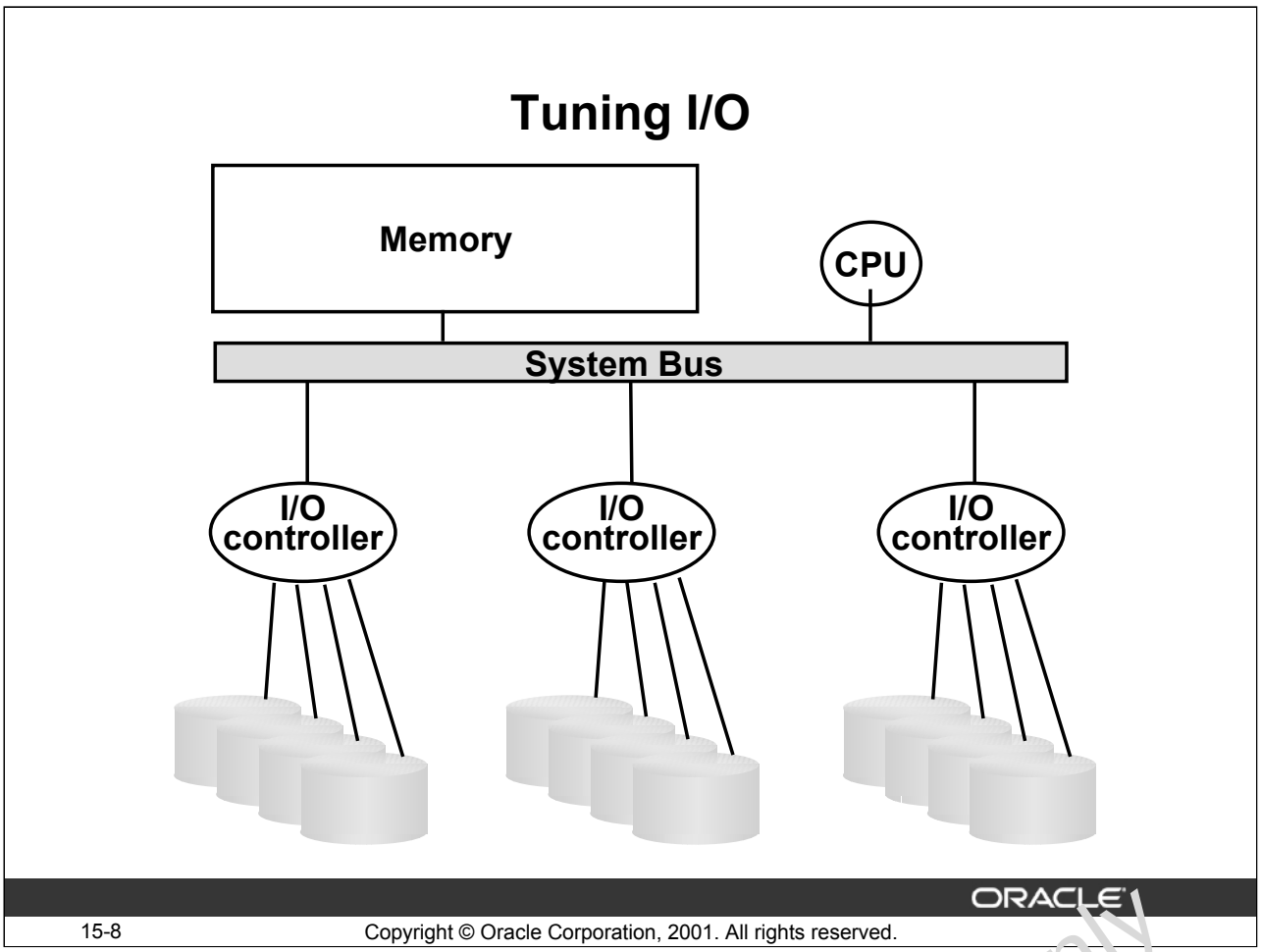
Besides tuning the SGA, the DBA can also affect paging and swapping performance in another way.

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

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.**

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 Nonblocking) I/O System Call

When Oracle issues an I/O request (read or write) using a asynchronous (or nonblocking) 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, plus you don't make the OS execute code to manage the file system cache or map file system blocks to physical disk blocks.

CPU Tuning

- **Guidelines:**
  - **Maximum CPU busy rate: 90%**
  - **Maximum OS/User processing ratio: 40/60**
  - **CPU load balanced across CPUs**
- **Monitoring:**
  - **CPU**
  - **Process**

ORACLE

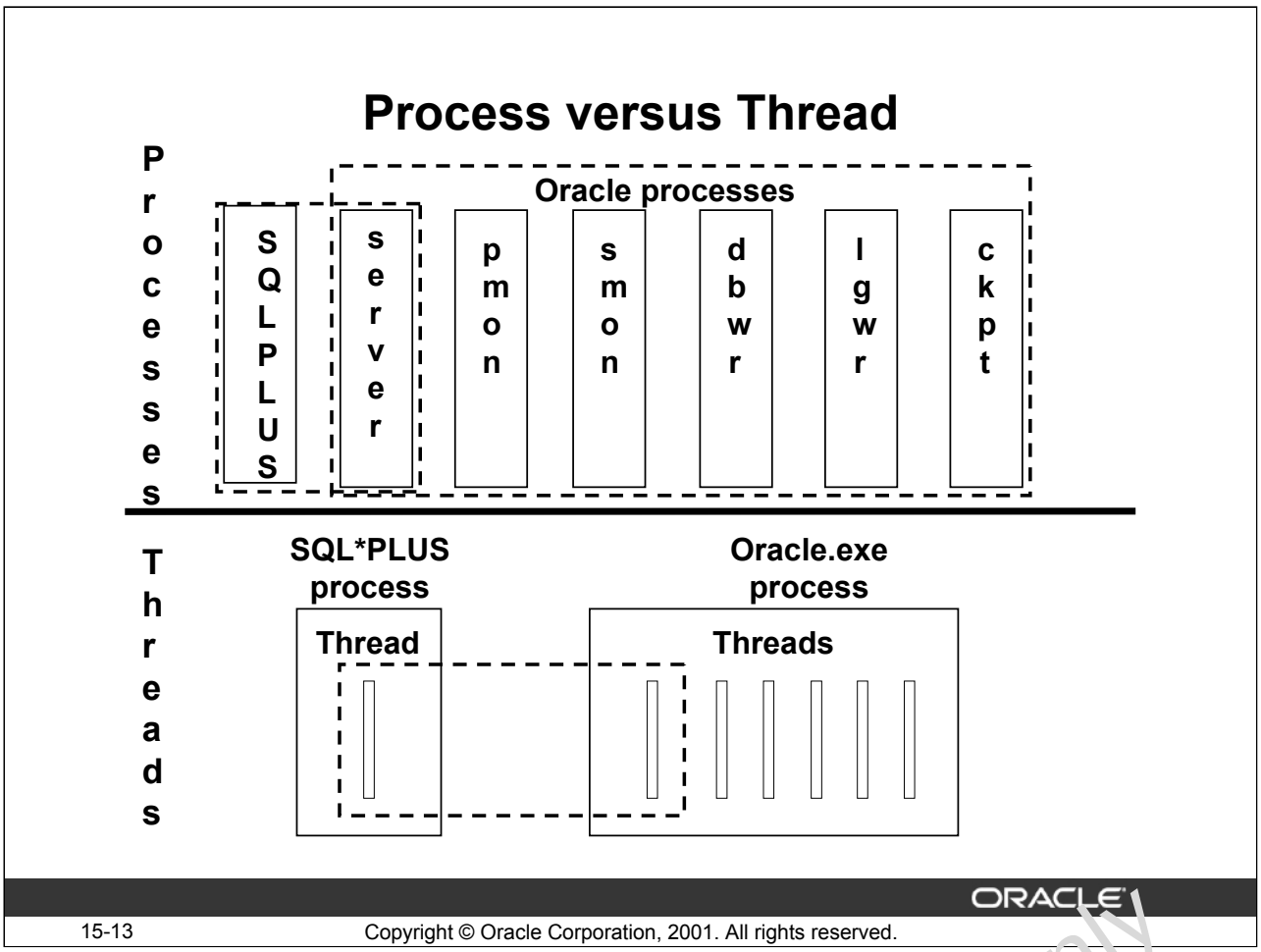
CPU Tuning Guidelines

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.



Processes and Threads

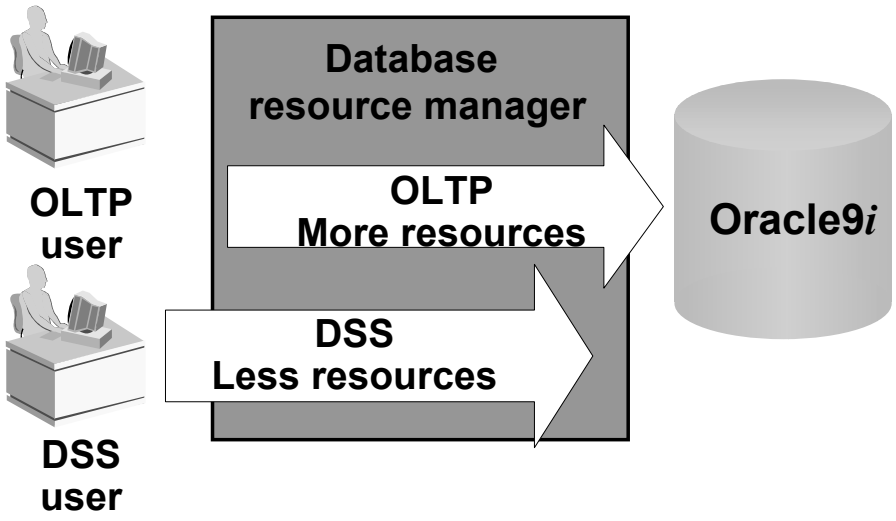
On most operating systems, each Oracle process is also an operating system process. However under some operating systems, notably Microsoft Windows NT and 2000, Oracle is implemented as a single operating system process with multiple threads. In Windows NT, Oracle processes threads share the memory allocated 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

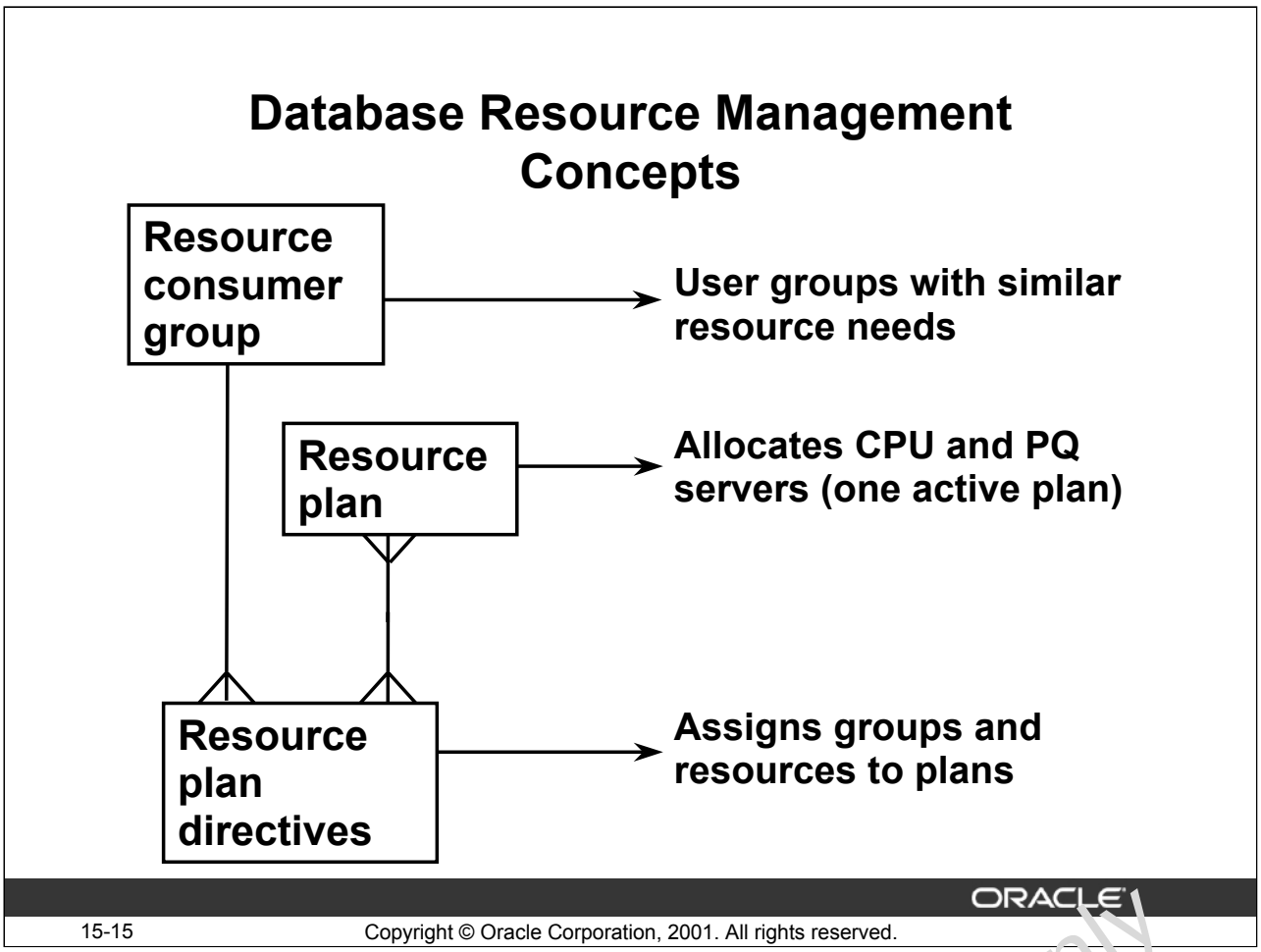


Overview

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 Oracle9*i* 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 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. 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. The 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

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 nonzero 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 throughput, then to prioritize among the consumer groups.

Distribution of Resources to Consumer Groups versus. 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			
	P1 CPU	P2 CPU	P3 CPU	P1 //
SYS_GROUP	100%	0%	0%	0
OTHER_GROUPS	0%	100%	0%	0
LOW_GROUP	0%	0%	100%	0

ORACLE

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

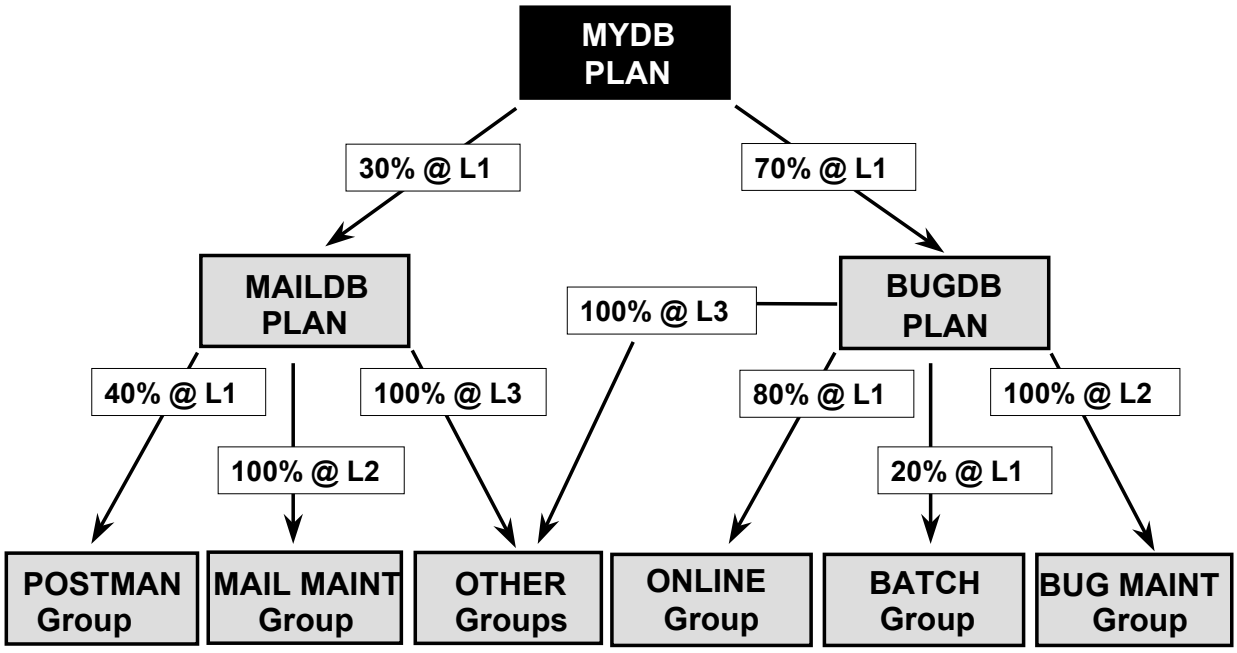
Provides a group with lower priority than SYS_GROUP and OTHER_GROUPS in this plan. You must decide which user sessions will be part of LOW_GROUP. Initially, no user is associated to this consumer group. Note that *switch* privilege is granted to PUBLIC for this group.

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.**

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.

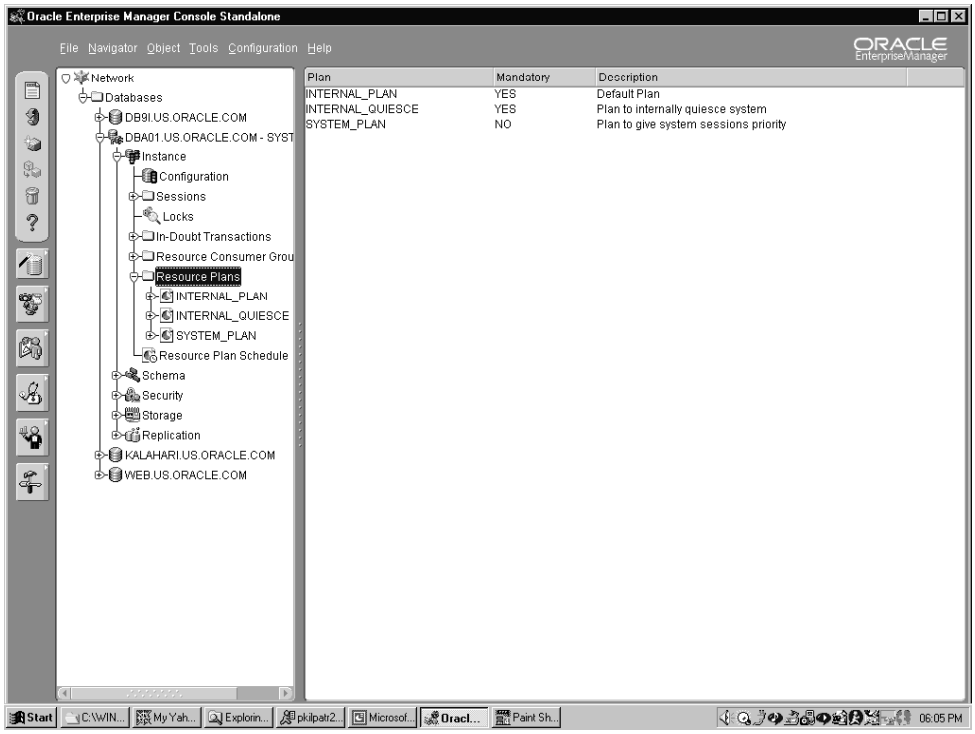
DBMS_RESOURCE_MANAGER_PRIVS Procedures

- 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, you will be looking at how to set up the Database Resource Manager by using the PL/SQL packages.

Enterprise Manager: Resource Manager



ORACLE

Enterprise Manager: Resource Manager

In the Enterprise Manager Console the DBA can administer resource plans.

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  );
```

ORACLE

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 `DBMS_RESOURCE_MANAGER` package .
- 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' );
```

ORACLE

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 `CREATE_PENDING_AREA` procedure. Views are available for inspecting all active resource plan as well as the pending areas (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);
```

ORACLE

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.

```
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);
```

Active Session Pool

The Database Resource Manager allows a DBA to limit the amount of concurrent active sessions per resource consumer group by defining an active session pool. Benefits of an active session pool:

- **Allows DBAs to meet performance service level objectives by limiting the concurrent system workload**
- **Reduces the number of servers taking resources in the system which avoid inefficient paging, swapping, and other resource depletion**

ORACLE

Active Session Pool

CPU resources are typically allocated at the beginning of a transaction or query and not freed until the transaction commits, or the query finishes. There is no way to free the resources associated with such operations until the operation completes. Thus, newly arriving operations either cause the system performance to be significantly impeded, or cause individual operations to error when unable to allocate a resource such as temporary space.

The active session pool feature allows the DBA to control the maximum number of concurrently active sessions per resource consumer group. With this functionality, a DBA can indirectly control the amount of resources that any resource consumer group uses since resource consumption is proportional to the number of active sessions. Once the active session pool is filled, the Database Resource Manager will queue all subsequent requests and run them only after the existing active sessions complete.

The main goal of the active session pool is to reduce the number of servers taking resources in the system, thus avoiding inefficient paging, swapping, and other resource depletion (memory, temporary space, and so on.) resulting from attempting to run too many jobs simultaneously. In essence, it is an attempt to guarantee some minimum resources to an operation, and to establish limits on the resource consumer group.

Active Session Pool Mechanism

- **The Active Session Pool**
  - **Size can be set per resource consumer group.**
  - **Only one size is allowed per consumer group.**
  - **Is the maximum number of concurrently active sessions.**
  - **Is defined as a session currently part of an active. transaction, query, or parallel operation.**
- **Once the active session pool is filled with active sessions, all subsequent sessions attempting to become active are queued.**
- **Individual parallel slaves do not count toward the number of sessions. The entire parallel operation counts as one active session.**

ORACLE

Active Session Pool Mechanism

Queuing Method

Once the active session pool is filled with active sessions, the Database Resource Manager queues all subsequent sessions, attempting to become active until other active sessions complete or become inactive. There is only one queue per resource consumer group and the queuing method will be one of first in first out (FIFO) with a time-out.

The Queue is a simply memory structure. There is no view that shows the queue directly. To see some queue information you can use the following views:

- V$SESSION - A new column called CURRENT_QUEUE_DURATION. If queued, it shows how long the session has been queued. It will be 0 (zero) if the session is not currently queued.
- V$RSRC_CONSUMER_GROUP - A new column called QUEUE_LENGTH. This shows the number of sessions currently queued per consumer group.

Active Session Pool Parameters

The active session pool is defined by setting the parameters:

- **ACTIVE_SESS_POOL_P1**
  - **Identifies the number of active sessions that establishes the resource consumer group's threshold.**
  - **Default is 1000000**
- **QUEUEING_P1**
  - **Indicates how long, in seconds, any session will wait on the queue before aborting the current operation**
  - **Default is 1000000**

ORACLE

Active Session Pool Parameters

ACTIVE_SESS_POOL_P1: Identifies the number of active sessions that establishes the resource consumer group's threshold, and therefore, its active session pool.

QUEUEING_P1: This optional parameter will indicate how long any session waits on the queue. If a session waits on the consumer group's queue for longer than the specified QUEUEING_P1, the operation will abort with an error.

Setting the Active Session Pool

Example:

GROUP	ACTIVE SESSION POOL
OLTP	
BATCH	ACTIVE_SESS_POOL_P1 = 5 QUEUEING_P1 = 600

OLTP: set no limit on concurrent active sessions

BATCH: set to limit concurrent active sessions to 5. QUEUEING_P1, set to 600, aborts all operations waiting on the queue for more than ten minutes.

ORACLE

Setting the Active Session Pool

Example

In the above example, the resource consumer group OLTP has no limit on the number of concurrent active sessions because the ACTIVE_SESS_POOL_P1 parameter was not set. The resource consumer group BATCH has an ACTIVE_SESS_POOL_P1 of SIZE 5. The BATCH group also has the QUEUEING_P1 parameter set to 600 (ten minutes). All sessions waiting on the queue for more than ten minutes will abort with an error.

Maximum Estimated Execution Time

- **The Database Resource Manager can estimate the execution time of an operation proactively.**
- **A DBA can specify a maximum estimated execution time for an operation at the resource consumer group level.**
  - **`MAX_ESTIMATED_EXEC_TIME`: Maximum estimated time an operation can take**
- **Operation will not start if the estimate is longer than `MAX_ESTIMATED_EXEC_TIME`**
- **The benefit of this feature is the elimination of the exceptionally large job that uses too many system resources.**
- **The Default is `1000000`.**

Maximum Estimated Execution Time

A DBA can define the maximum estimated execution time any operation can take at any given time by setting the resource plan directive `MAX_ESTIMATED_EXEC_TIME` parameter. After this parameter is set, the Database Resource Manager estimates the time a specific job will take. If the operation's estimate is more than the `MAX_ESTIMATED_EXEC_TIME` defined, then the operation will not start. This eliminates the exceptionally large job that would utilize too much of the systems resources.

If a resource consumer group has more than one plan directive referring to it, it may have more than one `MAX_ESTIMATED_EXEC_TIME`. The Database Resource Manager will then choose the most restrictive of all incoming values.

The estimated calculated time for a given statement is based on the statistics from the cost-based optimizer.

Automatic Consumer Group Switching

The Database Resource Manager automatically switches a session's consumer group based on the following resource plan directive parameters:

- **SWITCH_GROUP**
  - **Group switched to. Default is NULL.**
- **SWITCH_TIME**
  - **Active time in seconds. Default is 1000000.**
- **SWITCH_ESTIMATE**
  - **If value is TRUE, execution time estimate is used to decide whether to switch an operation even before it starts. Default is FALSE.**

This feature can be used to limit the resources consumed by long-running operations.

Automatic Change of Resource Manager Group

The Database Resource Manager will switch a running session to SWITCH_GROUP if the session is active for more than SWITCH_TIME seconds. Active means the session is running and consuming resources, not waiting idly for user input nor waiting for CPU cycles. After the session finishes its operation and becomes idle, it is switched back to its original group. If a resource consumer group has more than one plan directive referring to it, it may have more than one SWITCH_GROUP and SWITCH_TIME. The Database Resource Manager chooses the most restrictive of all incoming values.

If SWITCH_ESTIMATE is set to TRUE, the Database Resource Manager uses a predicted estimate of how long the operation will take to complete, to decide whether to switch a session before an operation even starts running. If this parameter is not set, the operation will just start running and will switch groups only if the other switch criteria are met.

A DBA can use this feature to manage the workload better by segregating long running batch jobs from short OLTP transactions. Batch jobs can be automatically assigned to consumer groups with lower resource allocation during day time to ensure good response time for OLTP users.

Undo Quota

New UNDO_POOL plan directive:

- **Limits the amount of undo space that can be used**
- **When exceeded, prevents DML**
- **SELECT statements are still allowed**
- **Is specified in kilobytes**
- **Default is 1000000**

Undo Quota

The Database Resource Manager now allows a DBA to manage the undo space consumed by long running transactions, using a resource plan directive parameter, UNDO_POOL.

It is defined as a quota of undo space per resource consumer group.

Whenever the total undo space is exceeded, no further INSERT, UPDATE, or DELETE will be allowed until undo space is freed by another session in the same group or undo quota is increased.

If the consumer group's quota is exceeded during the execution of a DML statement, the operation will abort and return an error.

The UNDO_POOL limit can be used with automatic undo management and user defined rollback segments.

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' );
```

ORACLE

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 );
```

ORACLE

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');
```

ORACLE

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            PRIVILEGE                   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**

ORACLE

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.**

ORACLE

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 to explain how:

- **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**

16

Workshop Overview

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 four 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

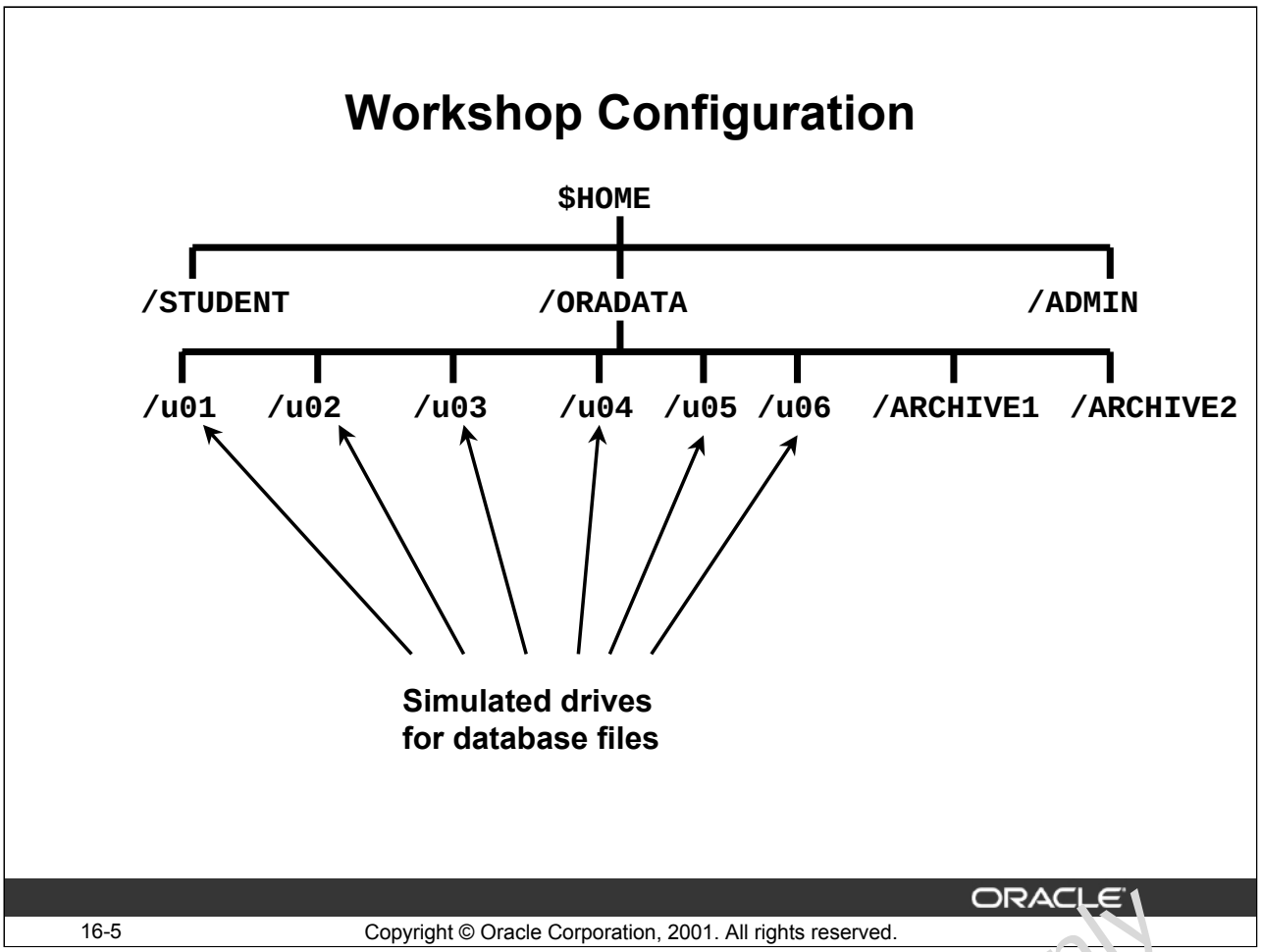
Present Situation

The company has two OLTP type users and two DSS type. The system was set up by a trainee DBA, and though it works, the performance is not acceptable. 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, what tables could do with an index, and so on.



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`.

ORACLE

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.**

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. The purpose of the workshop is to be as realistic as possible.

Workshop Scenarios

Choose from the following scenarios:

- **Shared pool**
- **Buffer cache**
- **Redo log buffer**
- **Indexes**
- **Rollback segments / undo tablespace**
- **Sort area size**
- **Reading STATSPACK report**

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 by 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, 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?**

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:**
  - **Collect and review statistics.**
  - **List the objectives for enhanced performance before modifications.**
  - **Modify the instance and the database.**
  - **Re-collect and review new statistics.**
  - **Compare the new results with the objectives.**
- **Implement Oracle architectural options for enhancing performance**

ORACLE

Practice Solutions Using SQL Plus

Practice 2

The goal of this practice is to familiarize yourself with the different methods of collecting statistical information. Throughout this practice Enterprise Manager can be used if desired. SQL Worksheet can be used instead of SQL*Plus, and there are many uses for the Enterprise Manager Console. (Solutions for Enterprise Manager can be found in Appendix B).

1. Log on as directed by the instructor. If the database is not already started, connect to SQL*Plus 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.

   ```
   SQL> show parameter timed_statistics
   ```

   If a value of true is return, continue to question 2. If a value of false is returned, then either edit the `init.ora` and add the parameter `TIMED_STATISTICS = TRUE`

   Or, set the parameter dynamically using

   ```
   SQL> alter system set timed_statistics = true;
   ```

2. Connect to SQL*Plus 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 `DBA_TABLES` dictionary view. 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.

   ```
   SQL> ALTER SESSION SET SQL_TRACE = TRUE;
   SQL> SELECT COUNT(*) FROM DBA_TABLES;
   SQL> ALTER SESSION SET SQL_TRACE = FALSE;
   ```

3. At the operating system level view the resulting trace file located in the directory set by `USER_DUMP_DEST.` Do not try to interpret the content of the trace file as this is the topic of a later lesson.

   ```
   $cd $HOME/ADMIN/UDUMP
   $ls –l
   -rw-r-----   1 user487    dba         4444 Apr 24
   22:28 U487_ora_3270 tr:
   ```

4. Open two sessions, 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 the `DBMS_SYSTEM.SET_SQL_TRACE_IN_SESSION` procedure. Get the user ID and SERIAL# from V$SESSION.

   From Session 1

   ```
   $ SQL*Plus hr/hr
   ```

   Change to Session 2

   ```
   $ SQL*Plus sys/oracle as sysdba
   SQL> select username, sid, serial# from v$session
   where username = 'HR';
   ```

Practice 2 (continued)

```
SQL> begin
     2> dbms_system.set_sql_trace_in_session
     3> (&SID,&SERIALNUM,TRUE);
     4> end;
     5> /
```

Change to Session 1

```
SQL> select * from employees;
```

Change to Session 2

```
SQL> begin
     2> dbms_system.set_sql_trace_in_session
     3> (&SID,&SERIALNUM,FALSE);
     4> end;
     5> /
```

5. Confirm that the trace file has been created in the directory set by USER_DUMP_DEST

```
$cd $HOME/ADMIN/UDUMP
$ls –l
-rw-r-----   1 dba01    dba         4444 Apr 24 22:28
dba01_ora_3270.trc
-rw-r-----   1 dba01    dba        15443 Apr 24 22:42
dba01_ora_3281.trc
```

6. Connect to SQL*Plus 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 100 MB, and be dictionary managed (this is not a requirement of STATSPACK, but will be used later in the course). Name the data file tools01.dbf and place it in the $HOME/ORADATA/u05 directory.

   Note: Dictionary managed is not the default.

```
SQL> connect sys/oracle as sysdba
SQL> create tablespace TOOLS
     2> data file '$HOME/ORADATA/u05/tools01.dbf'
size 100m
     3> extent management dictionary;
```

7. Confirm and record the amount of free space available within the TOOLS tablespace by querying the DBA_FREE_SPACE view.

```
SQL> select tablespace_name, sum (bytes)
    2> from dba_free_space
    3> where tablespace_name = 'TOOLS'
    4> group by tablespace_name;
```

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:

```
User's Default Tablespace = TOOLS
User's Temporary Tablespace = TEMP
SQL > @$HOME/STUDENT/LABS/spcreate.sql
```

Practice 2 (continued)

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 recorded in step 7 will be the space required for the initial installation of STATSPACK. Note that the amount of storage space required will increase in proportion to the amount of information stored within the STATSPACK tables, that is, the number of snapshots. Subtract the value received now, from the value received in step 7 to get the amount of space required in order to install STATSPACK
10. Manually collect current statistics using STATSPACK by running the `snap.sql` script located in $HOME/STUDENT/LABS. This will return the snap_id for the snapshot just taken, which should be recorded.
```
SQL > @$HOME/STUDENT/LABS/snap.sql
```
11. In order to have STATSPACK automatically collect statistics every three minutes execute the `spauto.sql` script located it your $HOME/STUDENT/LABS directory. Query the database to confirm that the job has been registered using the `user_jobs` view.
```
SQL > @$HOME/STUDENT/LABS/spauto.sql
SQL > select job, next_date, next_sec, last_sec
      2 > from user_jobs;
```
12. After waiting for a period in excess of three minutes query the `stats$snapshot` view in order to list what snapshots have been collected. There must be at least two snapshots before moving to the next step.
```
SQL> select snap_id, to_char(startup_time,' dd Mon "at"
HH24:mi:ss') instart_fm,
  2> to_char(snap_time,'dd Mon YYYY HH24:mi') snapdat,
snap_level "level"
  3> from stats$snapshot
  4> order by snap_id;
```
13. Once there are at least two snapshots, start to generate a report. This is performed using the `spreport.sql` script found in the $HOME/STUDENT/LABS directory. The script lists the snapshot options available, and then requests the beginning snap ID and the end snap ID. The user is then requested to give a filename for the report. It is often best left to the default.
```
SQL> @$HOME/STUDENT/LABS/spreport.sql
```
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.
```
$ vi sp_X_Y.lst
```
where X is the starting snapshot, and Y is the ending snapshot (this is true if the default report filename was used).
15. Connect to the database as a system administrator "sys/oracle as sysdba".
```
SQL> Connect sys/oracle as sysdba
```
16. Query the database in order to determine what system wait events have been registered since startup using v$system_event.
```
SQL> select event, total_waits, time_waited
    2> from v$system_event;
```

Practice 2 (continued)

Dynamic Performance Views

17. Determine if there are any sessions actually waiting for resources, using `v$session_wait.`

```
SQL> select sid, event, p1text, wait_time, state
   2 > from 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 `user_jobs` view to get the job number. Then execute `DBMS_JOB.REMOVE (<job number>);`

```
SQL> connect perfstat/perfstat
SQL> select job, log_user
     2> from user_jobs;
SQL> execute dbms_job.remove ($job);
```

19. Connect to your database using Enterprise Manager. The lecturer will supply the information required to connect to the Oracle Management Server. When connected use Enterprise Manager to explore the database. Examine items such as the number of tablespaces, users, and tables.
20. From Enterprise Manager load Oracle Expert, and create a new Tuning session. Limit the tuning scope to 'Check for Instance Optimizations.' This is done in order to reduce the time taken to collect information. Collect a new set of data. Do not implement the changes that Oracle Expert recommends, as this will be done during the course.

Practice 3

The objective of this practice is to use diagnostic tools to monitor and tune the shared pool. Throughout this practice Enterprise Manager can be used if desired. SQL Worksheet can be used instead of SQL*Plus, and there are many uses for the Enterprise Manager Console. (Solutions for Enterprise Manager can be found in Appendix B).

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

```
SQL> show parameter shared_pool
NAME                                 TYPE        VALUE
------------------------------------ ----------- ---------------
shared_pool_reserved_size            big integer 2516582
shared_pool_size                     big integer 50331648
```

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

```
SQL> connect perfstat/perfstat
SQL> @$HOME/STUDENT/LABS/snap.sql
```

3. To simulate user activity against the database open two operating system sessions. In session 1 connect as hr/hr and run the `$HOME/STUDENT/LABS/lab03_03_1.sql` script. In the second session connect as hr/hr and run the `$HOME/STUDENT/LABS/lab03_03_2.sql` script.

```
In session 1
SQL> connect hr/hr
SQL> @$HOME/STUDENT/LABS/lab03_03_1.sql
In session 2
SQL> connect hr/hr
SQL> @$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:

```
SQL> connect system/manager
SQL> select sum(pins), sum(reloads),
  2> sum(pins) * 100 / sum(pins+reloads) "Pin Hit%"
  3> from v$librarycache;
```

Practice 3 (continued)

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.

```
SQL> connect system/manager
SQL> SELECT SUM(getmisses), SUM(gets),
  2  SUM(getmisses )*100/SUM(gets)"MISS %"
  3  FROM v$rowcache;
```

   If `GETMISSES` are lower than 15% of the `GETS`, it is a good ratio.

6. Connect as perfstat/perfstat and run the `$HOME/STUDENT/LABS/snap.sql` script to collect a statistic snap shot and obtain the snapshot number. Record this number.

```
SQL> connect perfstat/perfstat
SQL> @$HOME/STUDENT/LABS/snap.sql
```

7. As user perfstat/perfstat obtain the statistics report between the two recorded snapshot IDs (from questions 2 and 6) by running the `$HOME/STUDENT/LABS/spreport.sql` script.

```
SQL> connect perfstat/perfstat
SQL> @$HOME/STUDENT/LABS/spreport.sql

  The following is an example of using a beginning
  snapshot_id of 3, and an ending snapshot_id of 5.
  Specify the Begin and End Snapshot Ids
  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  Enter value for begin_snap:   3

  Enter value for end_snap: 5
  End   Snapshot Id specified: 5


  Specify the Report Name
  ~~~~~~~~~~~~~~~~~~~~~~~
  The default report file name is sp_3_5.  To use this
  name,press <return> to continue, otherwise enter an
  alternative.
  Enter value for report_name:
```

Practice 3 (continued)

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%?

   Increase the SHARED_POOL_SIZE parameter.

9. Determine which packages, procedures, and triggers are pinned in the shared pool by querying `v$db_object_cache`.

```
SQL> select name,type,kept
  2> from v$db_object_cache
  3> where type IN
  4> ('PACKAGE', 'PROCEDURE', 'TRIGGER', 'PACKAGE
  BODY');
```

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 `DBMS_SHARED_POOL.KEEP` procedure , that is created by running the `$ORACLE_HOME/rdbms/admin/dbmspool.sql` script.

```
SQL> CONNECT sys/oracle AS SYSDBA
SQL> @?/rdbms/admin/dbmspool
SQL> execute DBMS_SHARED_POOL.KEEP('SYS.STANDARD');
SQL> select distinct owner, name
  2> from v$db_object_cache
  3> where kept='YES'
  4> and name like '%STAND%';
```

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'`

```
SQL> select a.name, b.value
  2> from v$statname a, v$mystat b
  3> where a.statistic# = b.statistic#
  4> and 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:

```
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#;
```

Practice 4

The objective of this practice is to use available diagnostic tools to monitor and tune the database buffer cache. Throughout this practice Enterprise Manager can be used if desired. SQL Worksheet can be used instead of SQL*Plus, and there are many uses for the Enterprise Manager Console. (Solutions for Enterprise Manager can be found in Appendix B).

1. Connect as perfstat/perfstat user, and run a statistic snapshot, and note the snapshot number. This can be performed by running the script file `$HOME/STUDENT/LABS/snap.sql`.
   ```
   SQL> connect perfstat/perfstat
   SQL> @$HOME/STUDENT/LABS/snap.sql
   ```
2. To simulate user activity against the database, connect as hr/hr user and run the `lab04_02.sql` script.
   ```
   SQL> connect hr/hr
   SQL> @$HOME/STUDENT/LABS/lab04_02.sql
   ```
3. Connect as `system/manager` and measure the hit ratio for the database buffer cache using the `v$sysstat` view. Determine if it is a good ratio or not.
   To query the V$SYSSTAT view:
   ```
   SQL> SELECT 1 - (phy.value – lob.value – dir.value)
        2                                / ses.value  "CACHE
     HIT RATIO“
        3>  FROM   v$sysstat ses, v$sysstat lob,
        4>         v$sysstat dir, v$sysstat phy
        5>  WHERE  ses.name = 'session logical reads‘
        6>  AND    dir.name = ‘physical reads direct‘
        7>  AND    lob.name = 'physical reads direct (lob)‘
        8>  AND    phy.name = 'physical reads';
   ```
4. Connect as perfstat/perfstat, and run a statistic snapshot, and note the snapshot number. This can be performed by running the `$HOME/STUDENT/LABS/snap.sql` script file.
   ```
   SQL> connect perfstat/perfstat
   SQL> @$HOME/STUDENT/LABS/snap.sql
   ```
5. Use the report from Statspack between the last two snapshots to check the buffer cache hit ratio, using the `$HOME/STUDENT/LABS/spreport.sql` script. Then analyze the buffer hit % in the “Instance Efficiency Percentages” section.
   ```
   SQL> @$HOME/STUDENT/LABS/spreport.sql
   ```
   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.

Practice 4 (continued)

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 temp_emps table.

```
SQL> connect system/manager
SQL> analyze table hr.temp_emps compute statistics;
SQL> select table_name , blocks
  2  from dba_tables
  3  where table_name in ('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 4 MB for the KEEP pool. This will generate an error due to insufficient memory in the SGA.

```
SQL> alter system set db_keep_cache_size=4M;
```

8. To resolve the memory problem reduce the size of the shared pool by 8 MB, 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 4 MB.

   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.

```
SQL> alter system set shared_pool_size=40M;
SQL> alter system set db_keep_cache_size=4M;
```

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.

```
SQL> connect system/manager
SQL> alter table hr.temp_emps
  2> storage (buffer_pool keep);
SQL> select table_name, buffer_pool
  2> FROM dba_tables
  3> WHERE buffer_pool = 'KEEP';
```

10. Connect as hr/hr, and run the $HOME/STUDENT/LABS/lab04_10.sql script. This will execute a query against the temp_emps table in the hr schema.

```
SQL> connect hr/hr
SQL> @$HOME/STUDENT/LABS/lab04_10.sql
```

11. Connect using sys/oracle as sysdba and check for the hit ratio in different buffer pools, using the V$BUFFER_POOL_STATISTICS view.

```
SQL> select name, physical_reads, db_block_gets,
  2> consistent_gets, 1 - (physical_reads
  3> / (db_block_gets + consistent_gets)) "hits"
  4> from v$buffer_pool_statistics;
```

Practice 5

Throughout this practice Enterprise Manager can be used if desired. SQL Worksheet can be used instead of SQL*Plus, and there are many uses for the Enterprise Manager Console. (Solutions for Enterprise Manager can be found in Appendix B).

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.

```
SQL> connect perfstat/perfstat
SQL> @$HOME/STUDENT/LABS/snap;
```

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.

```
SQL> connect sh/sh
SQL> @$HOME/STUDENT/LABS/lab05_02.sql
```

3. Connect as system/manager and query the `V$SYSSTAT` view to determine if there are space requests for the redo log buffer.

```
SQL> SELECT  RBAR.name, RBAR.value, RE.name, RE.value
  2> FROM     v$sysstat RBAR, v$sysstat RE
  3> WHERE    RBAR.name = 'redo buffer allocation
  retries'
  4> AND      RE.name   = 'redo entries';
```

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.

```
SQL> @$HOME/STUDENT/LABS/snap.sql;
SQL> @$HOME/STUDENT/LABS/spreport.sql
```

   - From the list of snapshots select a beginning and end value. The beginning should be the value recorded in step 1, and the end value step 4.
   - Note the name of the report file being generated.

5. Increase the size of the redo log buffer in the init.ora file located in the `$HOME/ADMIN/PFILE` directory
   This is performed by increasing the value of `LOG_BUFFER`.

Practice 6

Throughout this practice Enterprise Manager can be used if desired. SQL Worksheet can be used instead of SQL*Plus, and there are many uses for the Enterprise Manager Console. (Solutions for Enterprise Manager can be found in Appendix B).

1. Connect as system/manager and diagnose database file configuration by querying the `V$DATAFILE, V$LOGFILE` and `V$CONTROLFILE` dynamic performance views.

```
SQL> connect system/manager
SQL> SELECT name FROM v$data file
  2>  UNION
  3> SELECT member FROM v$logfile
  4>  UNION
  5> SELECT name FROM v$controlfile
  6>  UNION
  7> SELECT value from v$parameter
  8>  WHERE (name like 'log_archive_dest%'
  9>   and name NOT like 'log_archive_dest_state%')
 10>   or name in
 11>
  ('log_archive_dest','log_archive_duplex_dest');
```

2. Diagnose database file usage by querying the `V$FILESTAT` dynamic performance view, combine with `V$DATAFILE` in order to get the data file names

```
SQL> SELECT phyrds,phywrts,d.name
  2> FROM v$data file d, v$filestat f
  3> WHERE d.file#=f.file#;
```

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'

```
SQL> SELECT event, total_waits time_waited, average_wait
     2> from v$system_event
     3> where event = 'log file sync'
     4> or event = '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.

Practice 6 (continued)

4. Connect as perfstat/perfstat and diagnose file usage from STATSPACK.
   a. Generate a STATSPACK report using `$HOME/STUDENT/LABS/spreport.sql`.
   b. Locate and open the report file.
   c. Examine the report, and search for the string "File IO Stats"

   Note: On a production database care should 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 data file, 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 data file 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 `log_checkpoints_to_alert` parameter to TRUE using the "alter system set" command.

   ```
   SQL> alter system set log_checkpoints_to_alert = true;
   ```
6. Connect as sh/sh and execute the `$HOME/STUDENT/LABS/lab06_06.sql` script to provide a workload against the database.

   ```
   SQL> connect sh/sh
   SQL> @$HOME/STUDENT/LABS/lab06_06.sql
   ```
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

   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 of DML activity.

Practice 7

Throughout this practice Enterprise Manager can be used if desired. SQL Worksheet can be used instead of SQL*Plus, and there are many uses for the Enterprise Manager Console. (Solutions for Enterprise Manager can be found in Appendix B).

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.

```
SQL> connect system/manager
SQL> select name, value
  2> from   v$sysstat
  3> where  name like 'sorts%';
```

   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.

```
SQL> connect sh/sh
SQL> alter session set sort_area_size = 512;
SQL> @$HOME/STUDENT/LABS/lab07_02.sql;
```

   Note: If this script fails due to a lack of free space in the TEMP tablespace. Resize the temporary tablespace.

```
SQL> alter database tempfile
    2>  '$HOME/ORADATA/u02/temp01.dbf' resize 200m;
```

3. Connect as system/manger and query the TABLESPACE_NAME, CURRENT_USERS, USED_EXTENTS, and FREE_EXTENTS columns from the V$SORT_SEGMENT view. The USED_EXTENTS and FREE_EXTENTS columns are useful in monitoring the usage of the TEMPORARY tablespace.

```
SQL> select tablespace_name, current_users,
  used_extents, free_extents
    2> from v$sort_segment;
```

   Note: If this statement returns no rows, it means that all sort operations since startup h have completed in memory.

4. To decrease the sorts number of sorts going to a temporary tablespace, increase the value of the SORT_AREA_SIZE parameter to 512000 using the "alter session" command.

```
SQL> Alter session set Sort_area_size = 512000;
```

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)

```
SQL> connect system/manager
SQL> alter system set PGA_AGGREGATE_TARGET = 10M;
SQL> alter system set WORKAREA_SIZE_POLICY = AUTO;
```

Practice 9

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. Throughout this practice Enterprise Manager can be used if desired. SQL Worksheet can be used instead of SQL*Plus, and there are many uses for the Enterprise Manager Console. (Solutions for Enterprise Manager can be found in Appendix B).

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 `$HOME/ADMIN/PFILE` and comment out the following lines:
      ```
      undo_management = AUTO
      undo_tablespace = UNDOTBS
      ```
   c. Save the modifications and start up the database. Confirm that the `UNDO_MANAGEMENT` is `MANUAL`, and `UNDO_TABLESPACE` is null.
2. Connect sys/oracle as SYSDBA and create a new rollback segment tablespace `RBS_TEST` that is 2 MB in size using the `CREATE TABLESPACE` command. Name the data file `rbs_test.dbf` and place it in the `$HOME/ORADATA/u03` directory. Create the tablespace as dictionary managed.
   ```
   SQL> connect sys/oracle as sysdba
   SQL> CREATE TABLESPACE rbs_test
     2> DATAFILE '$HOME/ORADATA/u03/rbs_test.dbf' size 2M
     3> extent management dictionary;
   ```
   Note: This is *not* to be an UNDO tablespace, and you must specify that it is to be dictionary managed.
3. For the purposes of this practice, create a new rollback segment called RBSX in the `RBS_TEST` tablespace. For the storage parameters, use 64 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 1280 KB automatically.
   ```
   SQL> create rollback segment rbsx
     2> tablespace rbs_test
     3> storage (initial 64K next 64K minextents 20
     4> optimal 1280K);
   ```
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.
   ```
   SQL> alter rollback segment rbsx online;
   SQL> select segment_id,segment_name,status
     2> from dba_rollback_segs;
   ```

Practice 9 (continued)

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`.

```
SQL> select usn, writes
     2> from v$rollstat
     3>  where usn>0;
```

6. Open two sessions. In session 1 connect as hr/hr, and run the `$HOME/STUDENT/LABS/ins_temps.sql` script . The script inserts 100 new rows into the `TEMP_EMPS` table. Do *not* COMMIT this transaction. In the second session, log in as system/manager, determine how many rollback segment blocks or bytes the transaction is using. To do this query the writes column of `V$ROLLSTAT` to get the number of bytes written in the RBSX rollback segment so far. Record this value.

   In session 1:

```
SQL> connect hr/hr
SQL> @$HOME/STUDENT/LABS/ins_temps.sql
```

   In Session 2:

```
SQL> connect system/manager
SQL> select usn, writes
  2> from v$rollstat
  3> where usn>0;
   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.

```
SQL> SELECT s.username, t.used_ublk, t.start_time
  2> FROM v$transaction t, v$session s
  3>  WHERE t.addr = s.taddr;
```

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.

   In session 1:

```
SQL> commit;
SQL> @$HOME/STUDENT/LABS/del_temps.sql
In session 2:
SQL> select usn, writes
  2> from v$rollstat
  3> where usn>0;
```

Practice 9 (continued)

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.

```
SQL> select sum(waits)/sum(gets) "Ratio",
  2> sum(waits) "Waits", sum(gets) "Gets“
  3> from v$rollstat;
```

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.

```
SQL> select * from v$system_event
  2> where event ='undo segment tx slot';
```

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.

```
SQL> set transaction use rollback segment rbsx;
SQL> @$HOME/STUDENT/LABS/ins_temps.sql
SQL> select s.username, rn.name
  2> from v$session s, v$transaction t,
  3> v$rollstat r, v$rollname rn
  4> where s.saddr = t.ses_addr
  5> and   t.xidusn = r.usn
  6>  and r.usn = rn.usn;
```

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 located $HOME/ADMIN/PFILE and uncomment the following lines:
       undo_management = AUTO
       undo_tablespace = UNDOTBS
    c. Save the modifications and start up the database. Confirm that the UNDO_MANAGEMENT is AUTO, and UNDO_TABLESPACE is UNDOTBS

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). Throughout this practice Enterprise Manager can be used if desired. SQL Worksheet can be used instead of SQL*Plus, and there are many uses for the Enterprise Manager Console. (Solutions for Enterprise Manager can be found in Appendix B).

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.
```
SQL> connect hr/hr
SQL> UPDATE TEMP_EMPS
  2> SET SALARY = SALARY * 1.1
  3> where salary <15000;
```
2. In session 3 (sys/oracle as sysdba) check to see if any locks are being held by querying the V$LOCK.
```
SQL> connect sys/oracle as sysdba
SQL> select SID, TYPE, ID1, LMODE, REQUEST
  2> from v$lock
  3> where type in ('TX','TM');
```
3. In session 2 (user hr/hr – the session not yet used) drop the TEMP_EMPS table. Does it work?
```
SQL> connect hr/hr
SQL> DROP TABLE hr.temp_emps;
```
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.
```
SQL> connect hr/hr
SQL> UPDATE TEMP_EMPS
  2> SET SALARY = SALARY * 1.05
  3> where salary > 15000;
```
5. In session 3, check to see what kind of locks are being held on the TEMP_EMPS table, using the V$LOCK view.
```
SQL> SELECT sid, type, id1, id2, lmode, request
  2> FROM v$lock
  3> WHERE id1 =
  4> (SELECT object_id FROM dba_objects
  5> WHERE object_name = 'TEMP_EMPS'
  6> AND   object_type = 'TABLE');
```

Practice 10 (continued)

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.
   In session 2:

```
SQL> rollback;
SQL> UPDATE hr.temp_emps
  2  SET MANAGER_id = 10
  3  WHERE salary < 15000;
(This session will be hanging).
```

7. In session 3, check to see what kind of locks are being held on the `TEMP_EMPS` table, using the `V$LOCK` view.

```
SQL> SELECT sid, type, id1, id2, lmode, request
FROM v$lock
WHERE id1 = (SELECT object_id FROM dba_objects
              WHERE object_name = 'TEMP_EMPS'
              AND   object_type = 'TABLE');
```

8. In session 3, run the `$ORACLE_HOME/rdbms/admin/catblock.sql` script. The script will create a view, `DBA_WAITERS`, that gives information regarding sessions holding or waiting on a lock. Use this view to determine the session id for the session that is holding locks. Use this value to query `v$session` to get the serial number for the session holding the lock. Then issue the alter system kill session command in order to release the session holding the lock.

```
SQL>
SQL> @$ORACLE_HOME/rdbms/admin/catblock.sql
SQL> select waiting_session, holding_session
     2> from dba_waiters;
SQL> select SID,serial#,username
  2> from v$session
  3>  where SID ='<SID>';
SQL> ALTER SYSTEM KILL SESSION '<SID>,<SERIAL#>';
```

   Note: The second session would now show the blocking message.

Practice 12

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. Throughout this practice Enterprise Manager can be used if desired. SQL Worksheet can be used instead of SQL*Plus, and there are many uses for the Enterprise Manager Console. (Solutions for Enterprise Manager can be found in Appendix B).

1. Connect as hr/hr, and create the PLAN_TABLE under the HR schema, if it is not already created, by running $ORACLE_HOME/rdbms/admin/utlxplan.sql
   ```
   SQL> connect hr/hr
   SQL> @$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 $HOME/STUDENT/LABS/lab12_02.sql. View the generated plan by querying object name, operation, optimizer from PLAN_TABLE.
   ```
   SQL> alter session set optimizer_goal = rule;
   SQL> explain plan for
        2> @$HOME/STUDENT/LABS/lab12_02.sql
   SQL> select object_name, operation, optimizer
        2> 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 $HOME/STUDENT/LABS/lab12_02.sql. Notice that the optimizer mode, and the explain plan have changed.
   ```
   SQL> alter session set optimizer_goal = all_rows;
   SQL> explain plan for
        2> @$HOME/STUDENT/LABS/lab12_02.sql
   SQL> select object_name, operation, optimizer
        2> from plan_table;
   ```
   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 $HOME/STUDENT/LABS/lab12_04.sql script. Examine the script which is a copy of $HOME/STUDENT/LABS/lab12_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.
   ```
   SQL> truncate table plan_table;
   SQL> alter session set optimizer_goal = rule;
   ```

Practice 12 (continued)

```
SQL> explain plan for
     2> @$HOME/STUDENT/LABS/lab12_04.sql
SQL> select object_name, operation, optimizer
     2> from plan_table;
```

5. Exit out of SQL*Plus, change the directory to `$HOME/ADMIN/UDUMP` and delete all the trace files already generated.

```
SQL> exit
$ cd $HOME/ADMIN/UDUMP
$ rm *.trc
```

6. Connect as sh/sh and enable SQL Trace, using the alter session command, to collect statistics for the script, `$HOME/STUDENT/LABS/lab12_06.sql`. Run the script. After the script has completed, disable SQL Trace. and then format your trace file using `TKPROF`. Use the options `SYS=NO` and `EXPLAIN= sh/sh.` Name the file `myfile.txt.`

```
$ SQL*Plus sh/sh
SQL> alter session set sql_trace = true;
SQL> @$HOME/STUDENT/LABS/lab12_06.sql
SQL> alter session set sql_trace = false;
$ cd $HOME/ADMIN/UDUMP
$ ls -l
-rw-r----- 1 user457 dba 2180 May 4 00:27
  user457_ora_10424.trc
$ tkprof user457_ora_10424.trc myfile.txt explain=sh/sh
  sys=no
```

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.

```
$ more myfile.txt
```

Practice 12 (continued)

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.

```
$ SQL*Plus hr/hr
SQL> execute -
dbms_stats.create_stat_table('HR','MY_STATS');
```

   b. Save the current schema statistics into your local statistics table.

```
SQL> execute -
dbms_stats.export_schema_stats('HR','MY_STATS');
```

   c. Analyze all objects under the HR schema.

```
SQL> execute dbms_stats.gather_schema_stats('HR');
```

   d. Remove all schema statistics from the dictionary and restore the original statistics you saved in step b.

```
SQL> execute dbms_stats.delete_schema_stats('HR');
SQL> execute -
dbms_stats.import_schema_stats('HR','MY_STATS');
```

Practice 13

Throughout this practice Enterprise Manager can be used if desired. SQL Worksheet can be used instead of SQL*Plus, and there are many uses for the Enterprise Manager Console. (Solutions for Enterprise Manager can be found in Appendix B).

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.
```
SQL> connect / as sysdba
SQL> select tablespace_name, extent_management
     2> from dba_tablespaces;
```
2. Alter user HR to have the `TOOLS` tablespace as the default.
```
SQL> alter user hr default tablespace tools;
```
3. Examine the `v$system_event` view and note the total waits for the statistic enqueue.
```
SQL> select event, total_waits
     2> from v$system_event
     3> where event = 'enqueue';
```
Note: On a production system you would be more likely to pickup the contention through the `STATSPACK` report.
4. Also examine the `v$enqueue_stat` view for eq_type 'ST' in order to determine the total_wait# for the ST enqueue, which is the space management enqueue.
```
SQL> select * from v$enqueue_stat
     2>  where eq_type = 'ST';
```
5. Exit out of the SQL*Plus 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.
```
$ cd $HOME/STUDENT/LABS
$ ./lab13_04.sh
```
6. Connect as system/manager and again examine the `v$enqueue_stat` view for eq_type 'ST' in order to determine if the value of total_wait# for the ST enqueue, which is the space management enqueue.
```
$ SQL*Plus system/manager
SQL> select * from v$enqueue_stat
     2>  where eq_type = 'ST';
```
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).

Practice 13 (continued)

7. Create a new locally managed tablespace TEST, name the data file test01.dbf, and place it in the directory $HOME/ORADATA/u06. Set the size to 120 MB, and a uniform extent size of 20 KB.

```
SQL> create tablespace test
     2> data file '$HOME/ORADATA/u06/test01.dbf' size
  120M
     3> uniform size 20k;
```

8. Alter the default tablespace of user hr to test.

```
SQL> alter user hr default tablespace 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.

```
SQL> select * from v$enqueue_stat
     2>  where eq_type = 'ST';
```

10. Exit out of the SQL*Plus 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.

```
$ cd $HOME/STUDENT/LABS
$ ./lab13_04.sh
```

11. Again examine, and record the final total_wait# for 'ST' in the v$enqueue_stat view.

```
SQL> select * from v$enqueue_stat
     2>  where eq_type = 'ST';
```

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 because it is managing the space within the tablespace itself.

12. Connect as user hr/hr and run the $HOME/STUDENT/LABS/lab13_12.sql script. 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.

```
SQL> connect hr/hr
SQL> @$HOME/STUDENT/LABS/lab13_12.sql;
```

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.

```
SQL> analyze table new_emp compute statistics;
SQL> select table_name, chain_cnt
     2> from user_tables
     3> where table_name = 'NEW_EMP';
```

Practice 13 (continued)

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.

```
SQL> create index new_emp_name_idx on new_emp(last_name)
    2> tablespace indx;
SQL> select index_name, status
     2> from user_indexes
     3> where index_name = 'NEW_EMP_NAME_IDX';
```

15. Run the $HOME/STUDENT/LABS/lab13_15.sql script, 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 new_emp_name_idx index.

```
SQL> @$HOME/STUDENT/LABS/lab13_15.sql
SQL> analyze table new_emp compute statistics;
SQL> select table_name, chain_cnt
     2> from user_tables
     3> where table_name = 'NEW_EMP';
SQL> select index_name, status
     2> from user_indexes
     3> where index_name = '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 reanalyzing the new_emp table. 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.

```
SQL> alter table new_emp move
     2> tablespace users;
SQL> alter index new_emp_name_idx rebuild;
SQL> analyze table new_emp compute statistics;
SQL> select table_name, blocks, empty_blocks, chain_cnt
     2> from user_tables
     3> where table_name = 'NEW_EMP';
SQL> select index_name, status
     2> from user_indexes
     3> where index_name = 'NEW_EMP_NAME_IDX';
```

Practice 14

In this practice you will make use of the AUTOTRACE feature, and create the `plan_table` table. These are covered in detail in the chapter titled "SQL Statement Tuning." Throughout this practice Enterprise Manager can be used if desired. SQL Worksheet can be used instead of SQL*Plus, and there are many uses for the Enterprise Manager Console. (Solutions for Enterprise Manager can be found in Appendix B).

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 `employee_id` column the primary key, and name the primary key index `new_employees_employee_id_pk.`

```
SQL> connect hr/hr
SQL> create table new_employees
     2> (employee_id     number(6),
     3> first_name          varchar2(20),
     4> last_name           varchar2(25),
     5> email               varchar2(25),
     6> phone_number   varchar2(20),
     7> hire_date            date,
     8> job_id                 varchar2(10),
     9> salarly                 number(8,2),
    10> commission_pct number (2,2),
    11> manager_id        number(6),
    12> department_id     number(4),
    13> constraint new_employees_employee_id_pk
    14> primary key (employee_id))
    15> organization index;
```

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.

```
SQL> select table_name, iot_name, iot_type
            2> from user_tables
            3> where table_name like 'NEW_EMPLOYEES%';
```

   - The IOT is an index, and so will be found in the `user_indexes` view.

```
SQL> select index_name, index_type
2> from user_indexes
3> where table_name like 'NEW_EMPLOYEES%';
```

3. Populate the `new_employees` table with the rows from the `hr.employees` table

```
SQL> insert into new_employees
  2> select * from employees;
```

Practice 14 (continued)

4. Create a secondary B-Tree index on the `last_name` column 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.

```
SQL> create index last_name_new_employees_idx
     2> on new_employees(last_name)
     3> tablespace indx;
SQL> analyze index last_name_new_employees_idx
     2> compute statistics;
```

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.`

```
SQL> select index_name, index_type, blevel, leaf_blocks
     2> from user_indexes
     3> where index_name =
 'LAST_NAME_NEW_EMPLOYEES_IDX';
```

   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`.

```
SQL> create index emp_hist_dept_id_idx
     2> on employees_hist (department_id)
     3> tablespace indx
     4> reverse;
```

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.`

```
SQL> select index_name, index_type, blevel, leaf_blocks
     2> from user_indexes
     3> where index_name = 'EMP_HIST_DEPT_ID_IDX';
```

   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.`

```
SQL> create bitmap index bitmap_emp_hist_idx
     2> on employees_hist (job_id)
     3> tablespace indx;
```

Practice 14 (continued)

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.`

```
SQL> select index_name, index_type
     2> from user_indexes
     3> where index_name = 'BITMAP_EMP_HIST_IDX';
```

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.`

```
SQL> connect sh/sh
SQL> desc PLAN_TABLE
```

If the table is found:

```
SQL> truncate table plan_table;
```

If the table is not found then:

```
SQL> @$ORACLE_HOME/rdbms/admin/utlxplan
```

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.

```
SQL> connect sh/sh
SQL> create materialized view cust_sales
     2> enable query rewrite as
     3> select c.cust_last_name, sum(s.amount_sold)
     4> from sales s, customers c
     5> where c.cust_id = s.cust_id
     6> group by c.cust_last_name;
```

12. Confirm the creation of the materialized view `cust_sales` by querying the `USER_MVIEWS` data dictionary view, selecting the columns `mview_name, rewrite_enabled` and query.

```
SQL> select mview_name, rewrite_enabled, query
      2> from user_mviews;
```

Note: `rewrite_enabled` must be set to `Y` in order for the practice on query rewrite to work.

Practice 14 (continued)

13. Set AUTOTRACE to traceonly explain, to generate the explain plan for the query `$HOME/STUDENT/LABS/lab14_13.sql`

```
SQL> set autotrace traceonly explain
SQL> @$HOME/STUDENT/LABS/lab14_13.sql
```

14. Set the `query_rewrite_enabled` parameter to true for the session and run the same query, `$HOME/STUDENT/LABS/lab14_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.

```
SQL> alter session set query_rewrite_enabled = true;
SQL> @$HOME/STUDENT/LABS/lab14_13.sql
SQL> set autotrace off
SQL> alter session set query_rewrite_enabled = false;
```

B

Practice Solutions Using Enterprise Manager

Practice 2

The goal of this practice is to familiarize yourself with the different methods of collecting statistical information. Throughout this practice Enterprise Manager can be used if desired. SQL Worksheet can be used instead of SQL*Plus, and there are many uses for the Enterprise Manager Console.

1. Log on as directed by the instructor. If the database is not already started, connect to SQL*Plus 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.

   Use Enterprise Manager Console - Instance – Configuration

   Check 'All Initialization Parameters' Looking for 'TIMED_STATISTICS'

   Note: In order to restart the database using Enterprise Manager you will require SYSDBA privileges on the database.

2. Connect to SQL*Plus 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.

   Use Enterprise Manager – SQL Worksheet

   ```
   ALTER SESSION SET SQL_TRACE = TRUE;
   SELECT COUNT(*) FROM DBA_TABLES;
   ALTER SESSION SET SQL_TRACE = FALSE;
   ```

   Note: The trace file will be created on the same server as the database.

3. At the operating system level view the resulting trace file located in the directory set by `USER_DUMP_DEST.` Do not try to interpret the content of the trace file as this is the topic of a later lesson.

   ```
   $cd $HOME/ADMIN/UDUMP
   $ls –l
   -rw-r-----   1 user487    dba          4444 Apr 24
   22:28 U487_ora_3270.trc
   ```

4. Open two sessions, the first as user HR/HR, and the second as user "system/manager." From the second session generate a user trace file for the first session using `DBMS_SYSTEM.SET_SQL_TRACE_IN_SESSION` procedure. Get the user ID and SERIAL# from V$SESSION.

   Use Enterprise Manager – SQL Worksheet (open two for the two sessions)

   From Session 1

   ```
   connect hr/hr@<database>
   ```

   Change to Session 2

   ```
   connect system/manager@<database>
   select username, sid, serial# from v$session where
   username = 'HR';
   ```

Practice 2 (continued)

From SQL*Plus

```
SQL > begin
      2 > dbms_system.set_sql_trace_in_session
      3 > (&SID,&SERIALNUM,TRUE);
      4 > end;
      5 > /
```

Change to Session 1

```
select * from employees;
```

Change to SQL*Plus

```
SQL> begin
     2> dbms_system.set_sql_trace_in_session
     3> (&SID,&SERIALNUM,FALSE);
     4> end;
     5> /
```

5. Confirm that the trace file has been created in the directory set by USER_DUMP_DEST.

```
$cd $HOME/ADMIN/UDUMP
$ls –l
-rw-r-----   1 dba01    dba          4444 Apr 24 22:28
dba01_ora_3270.trc
-rw-r-----   1 dba01    dba         15443 Apr 24 22:42
dba01_ora_3281.trc
```

Note: Also view current user statistics through the Enterprise Manager Console.

6. Connect to SQL*Plus using "system/manager," and create a new tablespace (TOOLS) to hold the tables and other segments required by STATSPACK. This tablespace needs to be 100 MB, and be dictionary managed (this is not a requirement of STATSPACK, but 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.

   Use Enterprise Manager Console - Storage – Tablespace - Create

7. Confirm, and record, the amount of free space available within the TOOLS tablespace by querying the DBA_FREE_SPACE view.

   Use Enterprise Manager Console - Storage – Tablespace

8. Connect using "system/manager," 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:

   User's Default Tablespace = TOOLS

   User's Temporary Tablespace = TEMP

   Use SQL*Plus

```
SQL > @$HOME/STUDENT/LABS/spcreate.sql
```

   Note: Enterprise Manager – SQL Worksheet can be used, however, the file must be on the local workstation, and the directory must be provided by the instructor.

Practice 2 (continued)

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 recorded in step 7 will be the space required for the initial installation of STATSPACK. Note that the amount of storage space required will increase in proportion to the amount of information stored within the STATSPACK tables, that is, the number of snapshots.
   Use Enterprise Manager Console - Storage – Tablespace
   Note: You may need to refresh your screen
10. Manually collect current statistics using STATSPACK by running the snap.sql script located in $HOME/STUDENT/LABS. This will return the snap_id for the snapshot just taken, which should be recorded.
    Use SQL*Plus
    ```
    SQL > @$HOME/STUDENT/LABS/snap.sql
    ```
    Note: Enterprise Manager – SQL Worksheet can be used, however, the file must be on the local workstation, and the directory must be provided by the instructor.
11. In order to have STATSPACK automatically collect statistics every three minutes execute the spauto.sql script located it your $HOME/STUDENT/LABS directory. Query the database to confirm that the job has been registered using the user_jobs view.
    ```
    SQL > @$HOME/STUDENT/LABS/spauto.sql
    ```
    Note: Enterprise Manager – SQL Worksheet can be used, however, the file must be on the local workstation, and the directory must be provided by the instructor.
    Use Enterprise Manager – SQL Worksheet
    ```
    connect perfstat/perfstat@<database>
    select job, next_date, next_sec, last_sec
    from user_jobs;
    ```
12. After waiting for a period in excess of 3 minutes query the stats$snapshot view in order to list what snapshots have been collected. There must be at least two snapshots before moving to the next step.
    Use Enterprise Manager – SQL Worksheet
    ```
    select snap_id, to_char(startup_time,' dd Mon "at"
    HH24:mi:ss') instart_fm,
    to_char(snap_time,'dd Mon YYYY HH24:mi') snapdat,
    snap_level "level"
    from stats$snapshot
    order by snap_id;
    ```
13. Once there are at least two snapshots, start to generate a report. This is performed using the spreport.sql script found in the $HOME/STUDENT/LABS directory. The script list the snapshot options available, and then requests the beginning snap id and the end snap ID. The user is then requested to give a filename for the report. It is often best left to the default.
    Use SQL*Plus
    ```
    SQL> @$HOME/STUDENT/LABS/spreport.sql
    ```
    Note: Enterprise Manager – SQL Worksheet can be used, however, the file must be on the local workstation, and the directory must be provided by the instructor.

Practice 2 (continued)

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.

```
   $ vi sp_X_Y.lst
where X is the starting snapshot, and Y is the ending
   snapshot (this is true if the
default report filename was used).
```

15. Connect to the database as a system administrator "system/manager."
    Use Enterprise Manager – SQL Worksheet
    Connect system/manager
16. Query the database in order to determine what system wait events have been registered since startup using `v$system_event`.
    Use Enterprise Manager – SQL Worksheet

```
select event, total_waits, time_waited
from v$system_event;
```

Dynamic Performance Views

17. Determine if there are any sessions actually waiting for resources, using `v$session_wait`.
    Use Enterprise Manager – SQL Worksheet

```
select sid, event, p1text, wait_time, state
from 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>)`
    Use Enterprise Manager – SQL Worksheet

```
connect perfstat/perfstat@<database>
select job, log_user
from user_jobs;
execute dbms_job.remove (<job>);
```

Enterprise Manager

19. Connect to your database using Enterprise Manager. The lecturer will supply the information required to connect to the Oracle Management Server. When connected use Enterprise Manager to explore the database. Examine items such as the number of tablespaces, users, and tables.
19. From Enterprise Manager load Oracle Expert, and create a new Tuning session. Limit the tuning scope to "Check for Instance Optimizations." This is done in order to reduce the time taken to collect information. Collect a new set of data. Do not implement the changes that Oracle Expert recommends, as this will be done during the course.

Practice 3

The objective of this practice is to use diagnostic tools to monitor and tune the shared pool. Throughout this practice Enterprise Manager can be used if desired. SQL Worksheet can be used instead of SQL*Plus, and there are many uses for the Enterprise Manager Console. (Solutions for Enterprise Manager can be found in Appendix B).

1. Connect using "system/manager" and check the size of the shared pool.
   Use Enterprise Manager Console - Instance – Configuration
2. Connect as perfstat/perfstat user, execute the `$HOME/STUDENT/LABS/snap.sql` script to collect initial snapshot of statistics, and note the snapshot number by
   Use SQL*Plus
   ```
   SQL> connect perfstat/perfstat
   SQL> @$HOME/STUDENT/LABS/snap.sql
   ```
   Note: Enterprise Manager – SQL Worksheet can be used, however, the file must be on the local workstation, and the directory must be provided by the instructor.
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 $HOME/STUDENT/LABS/lab03_03_2.sql.
   Use SQL*Plus
   ```
   In session 1
      SQL> connect hr/hr
      SQL> @$HOME/STUDENT/LABS/lab03_03_1.sql
   ```
   Note: Enterprise Manager – SQL Worksheet can be used, however, the file must be on the local workstation, and the directory must be provided by the instructor.
   ```
   In session 2
      SQL> connect hr/hr
      SQL> @$HOME/STUDENT/LABS/lab03_03_2.sql
   ```
   Note: Enterprise Manager – SQL Worksheet can be used, however, the file must be on the local workstation, and the directory must be provided by the instructor.
4. Connect as "system/manager" and measure the pin-to-reload ratio for the library cache by querying v$librarycache. Determine if it is a good ratio or not.
   Use Enterprise Manager – SQL Worksheet
   ```
   connect system/manager@<database>
   select sum(pins), sum(reloads),
   sum(pins) * 100 /  sum(pins+reloads) "Pin Hit%"
   from v$librarycache;
   ```
   Or use Enterprise Manager – Performance Manager - Memory

Practice 3 (continued)

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.

   Use Enterprise Manager – SQL Worksheet

   ```
   connect system/manager@<database>
   SELECT SUM(getmisses), SUM(gets),
   SUM(getmisses )*100/SUM(gets)"MISS %"
   FROM v$rowcache;
   ```

   Or use Enterprise Manager – Performance Manager - Memory

   If `GETMISSES` are lower than 15% of the GETS, it is a good ratio.

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.

   Use SQL*Plus

   ```
   SQL> connect perfstat/perfstat
   SQL> @$HOME/STUDENT/LABS/snap.sql
   ```

   Note: Enterprise Manager – SQL Worksheet can be used, however, the file must be on the local workstation, and the directory must be provided by the instructor.

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.

   Use SQL*Plus

   ```
   SQL> connect perfstat/perfstat
   SQL> @$HOME/STUDENT/LABS/spreport.sql
   ```

   Note: Enterprise Manager – SQL Worksheet can be used, however, the file must be on the local workstation, and the directory must be provided by the instructor.

   ```
   The following is an example of using a beginning
   snapshot_id of 3, and an ending snapshot_id of 5.
   Specify the Begin and End Snapshot Ids
   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   Enter value for begin_snap:   3

   Enter value for end_snap: 5
   End   Snapshot Id specified: 5


   Specify the Report Name
   ~~~~~~~~~~~~~~~~~~~~~~~
   The default report file name is sp_3_5.  To use this
   name,press <return> to continue, otherwise enter an
   alternative.
   Enter value for report_name:
   ```

Practice 3 (continued)

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%?

   Increase the SHARED_POOL_SIZE parameter.

9. Determine which packages, procedures, and triggers are pinned in the shared pool by querying v$db_object_cache.

   Use Enterprise Manager – SQL Worksheet

```
select name,type,kept
from v$db_object_cache
 where type IN
 ('PACKAGE', 'PROCEDURE', 'TRIGGER', 'PACKAGE BODY');
```

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.

    Use SQL*Plus

```
SQL> CONNECT sys/oracle AS SYSDBA
SQL> @?/rdbms/admin/dbmspool
SQL> execute DBMS_SHARED_POOL.KEEP('SYS.STANDARD');
```

    Note: Enterprise Manager – SQL Worksheet can be used, however, the file must be on the local workstation, and the directory must be provided by the instructor.

    Use Enterprise Manager – SQL Worksheet

```
select distinct owner, name
from v$db_object_cache
where kept='YES' and name like '%STAND%';
```

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'

    Use Enterprise Manager – SQL Worksheet

```
select a.name, b.value
from v$statname a, v$mystat b
where a.statistic# = b.statistic#
and 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:

    Use Enterprise Manager – SQL Worksheet

```
select SUM(value)||' bytes' "Total session memory"
from v$sesstat, v$statname
where name = 'session uga memory'
and v$sesstat.statistic# = v$statname.statistic#;
```

Practice 4

The objective of this practice is to use available diagnostic tools to monitor and tune the database buffer cache. Throughout this practice Enterprise Manager can be used if desired. SQL Worksheet can be used instead of SQL*Plus, and there are many uses for the Enterprise Manager Console. (Solutions for Enterprise Manager can be found in Appendix B).

1. Connect as perfstat/perfstat user, and run a statistic snapshot, and note the snapshot number.

   Use SQL*Plus

   ```
   SQL> connect perfstat/perfstat
   SQL> @$HOME/STUDENT/LABS/snap.sql
   ```

   Note: Enterprise Manager – SQL Worksheet can be used, however, the file must be on the local workstation, and the directory must be provided by the instructor.

2. To simulate user activity against the database, connect as hr/hr user and run the `lab04_02.sql` script.

   Use SQL*Plus

   ```
   SQL> connect hr/hr
   SQL> @$HOME/STUDENT/LABS/lab04_02.sql
   ```

   Note: Enterprise Manager – SQL Worksheet can be used, however, the file must be on the local workstation, and the directory must be provided by the instructor.

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.

   Use Enterprise Manager – SQL Worksheet

   ```
   SELECT 1 - (phy.value – lob.value – dir.value) / ses.value  "HIT RATIO"
   FROM   v$sysstat ses, v$sysstat lob,  v$sysstat dir, v$sysstat phy
   WHERE  ses.name = 'session logical reads'
   AND dir.name = 'physical reads direct'
   AND lob.name = 'physical reads direct (lob)' AND phy.name = 'physical reads';
   ```

   Or use Enterprise Manager – Performance Manager - Memory

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.

   Use SQL*Plus

   ```
   SQL> connect perfstat/perfstat
   SQL> @$HOME/STUDENT/LABS/snap.sql
   ```

   Note: Enterprise Manager – SQL Worksheet can be used, however, the file must be on the local workstation, and the directory must be provided by the instructor.

Practice 4 (continued)

5. Use the report from Statspack between the last two snapshots to check the buffer cache hit ratio, using the script $HOME/STUDENT/LABS/spreport.sql. Then analyze the buffer hit % in the "Instance Efficiency Percentages" section.
   Use SQL*Plus
   ```
   SQL> @$HOME/STUDENT/LABS/spreport.sql
   ```
   Note: Enterprise Manager – SQL Worksheet can be used, however, the file must be on the local workstation, and the directory must be provided by the instructor.
   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 DBMS_STATS package, then query the BLOCKS column of the DBA_TABLES view for the table temp_emps.
   Use Enterprise Manager – SQL Worksheet
   ```
   connect system/manager
   exec dbms_stats.gather_table_stats('HR','TEMP_EMPS');
   select table_name , blocks
   from dba_tables
   where table_name in ('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.
   Use Enterprise Manager – SQL Worksheet
   ```
   alter system set db_keep_cache_size=4M;
   ```
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.
   Use Enterprise Manager – SQL Worksheet
   ```
   alter system set shared_pool_size=40M;
   alter system set db_keep_cache_size=4M;
   ```
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.
   Use Enterprise Manager – SQL Worksheet
   ```
   connect system/manager@<database>
   alter table hr.temp_emps storage (buffer_pool keep);
   select table_name, buffer_pool
   FROM dba_tables
   WHERE buffer_pool = 'KEEP';
   ```
   Note: In order to get the table hr.temp_emps into the keep pool, it might be required to bounce the database, and rerun the query.

Practice 4 (continued)

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.

    Use SQL*Plus

    ```
    SQL> connect hr/hr@<database>
    SQL> @$HOME/STUDENT/LABS/lab04_10.sql
    ```

    Note: Enterprise Manager – SQL Worksheet can be used, however, the file must be on the local workstation, and the directory must be provided by the instructor.

11. Connect using sys/oracle as sysdba and check for the hit ratio in different buffer pools, using the V$BUFFER_POOL_STATISTICS view.

    Use Enterprise Manager – SQL Worksheet

    connect system/manager@<database>

    ```
    select name, physical_reads, db_block_gets,
    consistent_gets, 1 - (physical_reads
    / (db_block_gets + consistent_gets)) "hits“
    from v$buffer_pool_statistics;
    ```

Practice 5

Throughout this practice Enterprise Manager can be used if desired. SQL Worksheet can be used instead of SQL*Plus, and there are many uses for the Enterprise Manager Console. (Solutions for Enterprise Manager can be found in Appendix B).

1. Connect as perfstat/perfstat and collect a snapshot of the current statistics by running the script $HOME/STUDENT/LABS/snap.sql. Record the snapshot id for later use.
   Use SQL*Plus
   ```
   SQL > connect perfstat/perfstat
     2 > @$HOME/STUDENT/LABS/snap;
   ```
   Note: Enterprise Manager – SQL Worksheet can be used, however, the file must be on the local workstation, and the directory must be provided by the instructor.
2. Connect as user SH/SH and run the $HOME/STUDENT/LABS/lab05_02.sql script in the STUDENT/LABS directory in order to have a workload.
   Use SQL*Plus
   ```
   SQL> connect sh/sh
   SQL> @$HOME/STUDENT/LABS/lab05_02.sql
   ```
   Note: Enterprise Manager – SQL Worksheet can be used, however, the file must be on the local workstation, and the directory must be provided by the instructor.
3. Connect as system/manager and query the V$SYSSTAT view to determine if there are space requests for the redo log buffer.
   Use Enterprise Manager – SQL Worksheet
   ```
   SELECT RBAR.name, RBAR.value, RE.name, RE.value
   FROM v$sysstat RBAR, v$sysstat RE
   WHERE RBAR.name = 'redo buffer allocation retries'
   AND RE.name = 'redo entries';
   ```
   Or use Enterprise Manager – Performance Manager – Wait Events
4. Connect as perfstat/perfstat and collect another set of statistics using the $HOME/STUDENT/LABS/snap.sql script. Then use $HOME/STUDENT/LABS/spreport.sql to generate a report using the two snapshot ids that you have collected. From the report determine log buffer statistics. View the generated file using an editor, and locate the "log buffer space" statistic.
   Use SQL*Plus
   ```
   SQL> @$HOME/STUDENT/LABS/snap.sql;
   SQL> @$HOME/STUDENT/LABS/spreport.sql
   ```
   Note: Enterprise Manager – SQL Worksheet can be used, however, the file must be on the local workstation, and the directory must be provided by the instructor.
   From the list of snapshots select a beginning and end value. The beginning should be the value recorded in step 1, and the end value step 4. Record the name of the report file being generated.
5. Increase the size of the redo log buffer in the init.ora file located in the $HOME/ADMIN/PFILE directory.
   This is performed by increasing the value of LOG_BUFFER.

Practice 6

Throughout this practice Enterprise Manager can be used if desired. SQL Worksheet can be used instead of SQL*Plus, and there are many uses for the Enterprise Manager Console. (Solutions for Enterprise Manager can be found in Appendix B).

1. Connect as system/manager and diagnose database file configuration by querying the V$DATAFILE, V$LOGFILE and V$CONTROLFILE dynamic performance views.
   Use Enterprise Manager – SQL Worksheet
   ```
   connect system/manager@<database>
   SELECT name FROM v$datafile
   UNION
   SELECT member FROM v$logfile
   UNION
   SELECT name FROM v$controlfile
   UNION
   SELECT value from v$parameter
   WHERE (name like 'log_archive_dest%'
   and name NOT like 'log_archive_dest_state%')
   or name in
   ('log_archive_dest','log_archive_duplex_dest');
   ```
   Or use Enterprise Manager – Performance Manager – I/O
2. Diagnose database file usage by querying the V$FILESTAT dynamic performance view, combine with V$DATAFILE in order to get the datafile names
   Use Enterprise Manager – SQL Worksheet
   ```
   SELECT phyrds,phywrts,d.name
   FROM v$datafile d, v$filestat f
   WHERE d.file#=f.file#;
   ```
   Or use Enterprise Manager – Performance Managern - I/O
3. Determine if there are waits for redo log files by querying the V$SYSTEM_EVENT dynamic performance view, where the waiting event is 'log file sync' or 'log file parallel write'
   Use Enterprise Manager – SQL Worksheet
   ```
   SELECT event, total_waits,time_waited, average_wait
   from v$system_event
   where event = 'log file sync'
   or event = 'log file parallel write';
   ```
   Or use Enterprise Manager – Performance Managern – Wait Events
4. 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.

Practice 6 (continued)

5. Connect as perfstat/perfstat and diagnose file usage from STATSPACK
   Use SQL*Plus
   a. Generate a Statspack report using $HOME/STUDENT/LABS/spreport.sql.
   b. Locate and open the report file.
   c. Examine the report, and search for the string "File IO Stats"
   Note: Enterprise Manager – SQL Worksheet can be used, however, the file must be on the local workstation, and the directory must be provided by the instructor.
   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.
6. 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.
   Use Enterprise Manager – SQL Worksheet
   ```
   alter system set log_checkpoints_to_alert = true;
   ```
   Or use Enterprise Manager Console - Instance
7. Connect as sh/sh and execute the $HOME/STUDENT/LABS/lab06_06.sql script to provide a workload against the database.
   Use SQL*Plus
   ```
   SQL> connect sh/sh
   SQL> @$HOME/STUDENT/LABS/lab06_06.sql
   ```
   Note: Enterprise Manager – SQL Worksheet can be used, however, the file must be on the local workstation, and the directory must be provided by the instructor.
8. 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
9. 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 of DML activity.

Practice 7

Throughout this practice Enterprise Manager can be used if desired. SQL Worksheet can be used instead of SQL*Plus, and there are many uses for the Enterprise Manager Console. (Solutions for Enterprise Manager can be found in Appendix B).

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.

   Use Enterprise Manager – SQL Worksheet

   connect system/manager@<database>

   ```
   select name, value
   from v$sysstat
   where name like 'sorts%';
   ```

   Or use Enterprise Manager – Performance Manager – Load

   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.

   Use Enterprise Manager – SQL Worksheet

   ```
   connect sh/sh
   alter session set sort_area_size = 512;
   ```

   Use SQL*Plus

   SQL> @$HOME/STUDENT/LABS/lab07_02.sql;

   Note: Enterprise Manager – SQL Worksheet can be used, however, the file must be on the local workstation, and the directory must be provided by the instructor.

   Note: If this script fails due to a lack of free space in the TEMP tablespace. Resize the temporary tablespace.

   Use Enterprise Manager – SQL Worksheet

   alter database tempfile '$HOME/ORADATA/u02/temp01.dbf' resize 200m;

   Or use Enterprise Manager Console – Storage - tablespaces

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.

   Use Enterprise Manager – SQL Worksheet

   select tablespace_name, current_users, used_extents, free_extents
   from v$sort_segment;

   Note: If this statement returns no rows, it means that all sort operations since startup have completed in memory.

Practice 7

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.
   Use Enterprise Manager – SQL Worksheet
   ```
   Alter session set Sort_area_size = 512000;
   ```
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)
   Use Enterprise Manager – SQL Worksheet
   ```
   connect system/manager
   alter system set PGA_AGGREGATE_TARGET = 10M;
   alter system set WORKAREA_SIZE_POLICY = AUTO;
   ```
   Or use Enterprise Manager Console – Instance - Configuration

Practice 9

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. Throughout this practice Enterprise Manager can be used if desired. SQL Worksheet can be used instead of SQL*Plus, and there are many uses for the Enterprise manager Console. (Solutions for Enterprise Manager can be found in Appendix B).

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.

   This can be performed using the Enterprise Manager Console. Use Enterprise Manager Console - Instance – Configuration in order to change the parameter value.

   Note: In order to restart the database using Enterprise Manager you will require SYSDBA privileges on the database
2. Connect sys/oracle as SYSDBA and create a new rollback segment tablespace RBS_TEST that is 2 MB in size using the CREATE TABLESPACE cammand. Name the datafile rbs_test.dbf and place it in the $HOME/ORADATA/u03 directory. Create the tablespace as Distionary Managed.

   Use Enterprise Manager Console - Storage – Tablespace – Create

   Note: This is NOT to be an UNDO Tablespace, and you must specify that it is to be Dictionary Managed.
3. For the purposes of this practice, create a new rollback segment called RBSX in the RBS_TEST tablespace. For the storage parameters, use 64 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 1280 KB automatically.

   Use Enterprise Manager Console - Storage –Rollback Segments – Create
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.

   Use Enterprise Manager Console - Storage –Rollback Segments
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

   Use Enterprise Manager – SQL Worksheet

   ```
   connect system/manager@<database>
   select usn, writes
   from v$rollstat
   where usn>0;
   ```

Practice 9 (continued)

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.

   In session 1 use SQL*Plus:

   SQL> connect hr/hr
   SQL> @$HOME/STUDENT/LABS/ins_temps.sql

   In Session 2 use Enterprise Manager – SQL Worksheet:

   connect system/manager@<database>

   ```
   select usn, writes
   from v$rollstat
   where usn>0;
   ```

   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.

   Use Enterprise Manager – SQL Worksheet

   ```
   SELECT s.username, t.used_ublk, t.start_time
   FROM v$transaction t, v$session s
   WHERE t.addr = s.taddr;
   ```

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.

   In session 1 (still using SQL*Plus):

   ```
   SQL> commit;
   SQL> @$HOME/STUDENT/LABS/del_temps.sql
   ```

   In session 2 (still using Enterprise Manager – SQL Worksheet):

   ```
   SELECT usn, writes
   from v$rollstat
   where usn>0;
   ```

Practice 9 (continued)

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.

   Use Enterprise Manager – SQL Worksheet

```
SELECT sum(waits)/sum(gets) "Ratio",
sum(waits) "Waits", sum(gets) "Gets“
from v$rollstat;
```

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.

    Use Enterprise Manager – SQL Worksheet

```
SELECT * from v$system_event
where event ='undo segment tx slot';
```

    Or use Enterprise Manager – Performance Manager – Wait Events

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.

    Use SQL*Plus:

```
SQL> set transaction use rollback segment rbsx;
SQL> @$HOME/STUDENT/LABS/ins_temps.sql
SQL> select s.username, rn.name
  2> from v$session s, v$transaction t,
  3> v$rollstat r, v$rollname rn
  4> where s.saddr = t.ses_addr
  5> and   t.xidusn = r.usn
  6>  and r.usn = rn.usn;
```

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

    This can be performed using the Enterprise Manager Console. Use Enterprise Manager Console - Instance – Configuration in order to change the parameter value.

    Note: In order to restart the database using Enterprise Manager you will require SYSDBA privileges on the database

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). Throughout this practice Enterprise Manager can be used if desired. SQL Worksheet can be used instead of SQL*Plus, and there are many uses for the Enterprise Manager Console. (Solutions for Enterprise Manager can be found in Appendix B).

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.
   Use Enterprise Manager – SQL Worksheet
   ```
   connect hr/hr@<database>
   UPDATE TEMP_EMPS
   SET SALARY = SALARY * 1.1
   where salary <15000;
   ```
2. In session 3 (system/manager) check to see if any locks are being held by querying the V$LOCK.
   Use Enterprise Manager – SQL Worksheet
   ```
   connect system/manager@<database>
   SELECT sid, type, id1, lmode, request
   FROM v$lock
   WHERE type in ('TX','TM');
   ```
3. In session 2 (user hr/hr – the session not yet used) drop the TEMP_EMPS table. Does it work?
   Use Enterprise Manager Console – Schema – Table – User - Remove
   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.
   Use Enterprise Manager – SQL Worksheet
   ```
   connect hr/hr@<database>
   UPDATE TEMP_EMPS
   SET SALARY = SALARY * 1.05
   where salary > 15000;
   ```
5. In session 3, check to see what kind of locks are being held on the TEMP_EMPS table, using the V$LOCK view.
   Use Enterprise Manager – Lock Monitor

Practice 10 (continued)

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.
   In session 2:
   Use Enterprise Manager – SQL Worksheet

```
rollback;
UPDATE hr.temp_emps
SET MANAGER_id = 10
WHERE salary < 15000;
```

   (This session will be hanging).
7. In session 3, check to see what kind of locks are being held on the TEMP_EMPS table, using the V$LOCK view.
   Use Enterprise Manager – Lock Monitor
8. In session 3, 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.
   In SQL*Plus:

```
SQL> connect sys/oracle
SQL> @$ORACLE_HOME/rdbms/admin/catblock.sql
SQL> select waiting_session, holding_session
  2> from dba_waiters;
SQL> select SID,serial#,username
  2> from v$session
  3>  where SID ='<SID>';
SQL> ALTER SYSTEM KILL SESSION '<SID>,<SERIAL#>';
```

   Note: The second session would now show the blocking message.
   OR
   Use Enterprise Manager – Lock Monitor to determine the blocking session, then use Enterprise Manager Console – Instance – Sessions

Practice 12

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. Throughout this practice Enterprise Manager can be used if desired. SQL Worksheet can be used instead of SQL*Plus, and there are many uses for the Enterprise Manager Console. (Solutions for Enterprise Manager can be found in Appendix B).

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

   Use SQL*Plus

   ```
   SQL> connect hr/hr
   SQL> @$ORACLE_HOME/rdbms/admin/utlxplan.sql
   ```

   Note: If plan_table already exists and holds rows truncate the table.

   Note: Enterprise Manager – SQL Worksheet can be used, however, the file must be on the local workstation, and the directory must be provided by the instructor.

2. Set the Optimizer_goal to rule based using the alter session command, and generate the explain plan for the statement $HOME/STUDENT/LABS/lab12_02.sql. View the generated plan by querying object_name, operation, optimizer from PLAN_TABLE.

   In SQL*Plus

   ```
   SQL> alter session set optimizer_goal = rule;
   SQL> explain plan for
     2> @$HOME/STUDENT/LABS/lab12_02.sql
   SQL> select object_name, operation, optimizer
     2> from plan_table;
   ```

   Note: Enterprise Manager – SQL Worksheet can be used, however, the file must be on the local workstation, and the directory must be provided by the instructor.

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/lab12_02.sql. Notice that the Optimizer mode, and the explain plan have changed.

   Use SQL*Plus

   ```
   SQL> alter session set optimizer_goal = all_rows;
   SQL> explain plan for
     2> @$HOME/STUDENT/LABS/lab12_02.sql
   SQL> select object_name, operation, optimizer
     2> from plan_table;
   ```

   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.

Practice 12 (continued)

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/lab12_04.sql. Examine the script which is a copy of $HOME/STUDENT/LABS/lab12_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.

   Use SQL*Plus

   ```
   SQL> truncate table plan_table;
   SQL> alter session set optimizer_goal = rule;
   SQL> explain plan for
     2> @$HOME/STUDENT/LABS/lab12_04.sql
   SQL> select object_name, operation, optimizer
     2> from plan_table;
   ```

5. Exit out of SQL*Plus, change the directory to $HOME/ADMIN/UDUMP and delete all the trace files already generated.

   ```
   SQL> exit
   $ cd $HOME/ADMIN/UDUMP
   $ rm *.trc
   ```

6. Connect as sh/sh and enable SQL TRACE, using the alter session command, to collect statistics for the script, $HOME/STUDENT/LABS/lab12_06.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

   ```
   $ SQL*Plus sh/sh
   SQL> alter session set sql_trace = true;
   SQL> @$HOME/STUDENT/LABS/lab12_06.sql
   SQL> alter session set sql_trace = false;
   $ cd $HOME/ADMIN/UDUMP
   $ ls -l
   -rw-r----- 1 user457 dba 2180 May 4 00:27 user457_ora_10424.trc
   $ tkprof user457_ora_10424.trc myfile.txt explain=sh/sh sys=no
   ```

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.

   ```
   $ more myfile.txt
   ```

Practice 12 (continued)

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.

   Use Enterprise Manager – SQL Worksheet

   ```
   Connect hr/hr@<database>
   exec dbms_stats.create_stat_table('HR','MY_STATS');
   ```

   b. Save the current schema statistics into your local statistics table.

   Use Enterprise Manager – SQL Worksheet

   ```
   exec dbms_stats.export_schema_stats('HR','MY_STATS');
   ```

   c. Analyze all objects under the HR schema.

   Use Enterprise Manager – SQL Worksheet

   ```
   exec dbms_stats.gather_schema_stats('HR');
   ```

   d. Remove all schema statistics from the dictionary and restore the original statistics you saved in step b.

   Use Enterprise Manager – SQL Worksheet

   ```
   exec dbms_stats.delete_schema_stats('HR');
   exec dbms_stats.import_schema_stats('HR','MY_STATS');
   ```

   Statistics can also be generated by:

   Using Enterprise Manager Console – Schema – Table - Analyze

Practice 13

Throughout this practice Enterprise Manager can be used if desired. SQL Worksheet can be used instead of SQL*Plus, and there are many uses for the Enterprise Manager Console. (Solutions for Enterprise Manager can be found in Appendix B).

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.
   Use Enterprise Manager Console - Tablespaces
2. Alter user HR to have the TOOLS tablespace as the default.
   Use Enterprise Manager Console – Security - Users
3. Examine the v$system_event view and note the total_waits for the statistic enqueue.
   Use Enterprise Manager – SQL Worksheet
   SQL> select event, total_waits
   2> from v$system_event
   3> where event = 'enqueue';
   Or use Enterprise Manager – Performance Manager – Wait Events

   Note: On a production system you would be more likely to pickup the contention through the STATSPACK report.
4. Also examine the view v$enqueue_stat for eq_type 'ST' in order to determine the total_wait# for the ST enqueue, which is the space management enqueue.
   Use Enterprise Manager – SQL Worksheet
   SQL> select * from v$enqueue_stat
   2> where eq_type = 'ST';
   Or use Enterprise Manager – Performance Manager – Wait Events
5. Exit out of the SQL*Plus 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.
   $ cd $HOME/STUDENT/LABS
   $ ./lab13_04.sh
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.
   Use Enterprise Manager – SQL Worksheet
   $ SQL*Plus system/manager
   SQL> select * from v$enqueue_stat
   2> where eq_type = 'ST';
   Or use Enterprise Manager – Peformance Manager – Wait Events

   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).

Practice 13 (continued)

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.
   Use Enterprise Manager Console – Storage - Tablespace - Create
8. Alter the default tablespace of user hr to test.
   Use Enterprise Manager – Security - Users

   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$enqueue_stat view.
   Use Enterprise Manager – SQL Worksheet
   ```
   SQL> select * from v$enqueue_stat
     2>  where eq_type = 'ST';
   ```
   Or use Enterprise Manager – Performance Manager – Wait Events
10. Exit out of the SQL*Plus 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.
    ```
    $ cd $HOME/STUDENT/LABS
    $ ./lab13_04.sh
    ```
11. Again examine, and record the final total_wait# for 'ST' in the v$enqueue_stat view.
    Use Enterprise Manager – SQL Worksheet
    ```
    SQL> select * from v$enqueue_stat
      2>  where eq_type = 'ST';
    ```
    Or use Enterprise Manager – Performance Manager – Wait Events

    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.
12. Connect as user hr/hr and run the script $HOME/STUDENT/LABS/lab13_12.sql. This will create a similar table (NEW_EMP) as the employees table but with PCTFREE = 0. The table is then populated with data from the employees table.
    Use SQL*Plus
    ```
    SQL> connect hr/hr
    SQL> @$HOME/STUDENT/LABS/lab13_12.sql;
    ```

Practice 13 (continued)

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.

    Use Enterprise Manager – SQL Worksheet

    ```
    SQL> analyze table new_emp compute statistics;
    SQL> select table_name, chain_cnt
      2> from user_tables
      3> where table_name = 'NEW_EMP';
    ```

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.

    Use Enterprise Manager Console – Schema – Indexes - Create

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.

    Use SQL*Plus

    ```
    SQL> @$HOME/STUDENT/LABS/lab13_15.sql
    SQL> analyze table new_emp compute statistics;
    SQL> select table_name, chain_cnt
      2> from user_tables
      3> where table_name = 'NEW_EMP';
    SQL> select index_name, status
      2> from user_indexes
      3> where index_name = '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.

    Use Enterprise Manager – SQL Worksheet

    ```
    alter table new_emp move tablespace users;
    alter index new_emp_name_idx rebuild;
    analyze table new_emp compute statistics;
    select table_name, blocks, empty_blocks, chain_cnt
    from user_tables where table_name = 'NEW_EMP';
    select index_name, status
    from user_indexes where index_name = 'NEW_EMP_NAME_IDX';
    ```

Practice 14

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". Throughout this practice Enterprise Manager can be used if desired. SQL Worksheet can be used instead of SQL*Plus, and there are many uses for the Enterprise Manager Console. (Solutions for Enterprise Manager can be found in Appendix B).

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

   Use Enterprise Manager – SQL Worksheet

   ```
   connect hr/hr@<database>
   create table new_employees
   2> (employee_id     number(6),
   3> first_name       varchar2(20),
   4> last_name        varchar2(25),
   5> email            varchar2(25),
   6> phone_number     varchar2(20),
   7> hire_date        date,
   8> job_id           varchar2(10),
   9> salarly          number(8,2),
   10> commission_pct number (2,2),
   11> manager_id     number(6),
   12> department_id  number(4),
   13> constraint new_employees_employee_id_pk
   14> primary key (employee_id))
   15> organization index;
   ```

   Or use Enterprise Manager Console – Schema - Table

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.
   - Use Enterprise Manager – SQL Worksheet

     ```
     select table_name, iot_name, iot_type from user_tables
     where table_name like 'NEW_EMPLOYEES%';
     ```

   - The IOT is an index, and so will be found in the user_indexes view.

     ```
     select index_name, index_type
     from user_indexes
     where table_name like 'NEW_EMPLOYEES%';
     ```

3. Populate the new_employees table with the rows from the hr.employees table

   Use Enterprise Manager – SQL Worksheet

   ```
   insert into new_employees
   select * from employees;
   ```

Practice 14 (continued)

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.

   Use Enterprise Manager Console – Schema – Index – Create
   Use Enterprise Manager Console – Schema – Index - Analyze

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.

```
SQL> select index_name, index_type, blevel, leaf_blocks
  2> from user_indexes
  3> where index_name = 'LAST_NAME_NEW_EMPLOYEES_IDX';
```

   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**.**

```
SQL> create index emp_hist_dept_id_idx
  2> on employees_hist (department_id)
  3> tablespace indx
  4> reverse;
```

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.

```
SQL> select index_name, index_type, blevel, leaf_blocks
  2> from user_indexes
  3> where index_name = 'EMP_HIST_DEPT_ID_IDX';
```

   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.

```
SQL> create bitmap index bitmap_emp_hist_idx
  2> on employees_hist (job_id)
  3> tablespace indx;
```

Practice 14 (continued)

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.
   Use Enterprise Manager Console – Schema - Index
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 $ORACLE_HOME/rdbms/admin/utlxplan.sql.
   Use Enterprise Manager – SQL Worksheet
      connect sh/sh@<database>
      desc PLAN_TABLE
   Or use Enterprise Manager Console – Schema - Table
   If the table is found:
   Use Enterprise Manager – SQL Worksheet
      truncate table plan_table;
   If the table is not found then:
   Use SQL*Plus
      SQL> @$ORACLE_HOME/rdbms/admin/utlxplan
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.
   Use Enterprise Manager – SQL Worksheet
      connect sh/sh@<database>
      create materialized view cust_sales
      enable query rewrite as
      select c.cust_last_name, sum(s.amount_sold)
      from sales s, customers c
      where c.cust_id = s.cust_id
      group by c.cust_last_name;
   Or use Enterprise Manager Console – Schema – Materialized View - Create
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.
   Use Enterprise Manager – SQL Worksheet
      SQL> select mview_name, rewrite_enabled, query from user_mviews;
   Or use Enterprise Manager Console – Schema – Materialized View

   Note: Rewrite_enabled must be set to Y in order for the practice on query rewrite to work.

Practice 14 (continued)

13. Set AUTOTRACE to traceonly explain, to generate the explain plan for the query $HOME/STUDENT/LABS/lab14_13.sql

    Use SQL*Plus

```
SQL> set autotrace traceonly explain
SQL> @$HOME/STUDENT/LABS/lab14_13.sql
```

14. Set the query_rewrite_enabled parameter to true for the session and run the same query, $HOME/STUDENT/LABS/lab14_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.

    Use SQL*Plus

```
SQL> alter session set query_rewrite_enabled = true;
SQL> @$HOME/STUDENT/LABS/lab14_13.sql
SQL> set autotrace off
SQL> alter session set query_rewrite_enabled = false;
```

C

Tuning Workshop

Workshop Scenarios

- **Small shared pool**
- **Small database buffer cache**
- **Small redo log buffer cache**
- **Missing indexes**
- **Rollback segments and undo tablespace**
- **Sort area size incorrectly set**
- **Reading STATSPACK report**

ORACLE

Workshop Scenario

At the beginning of each scenario, the database is shut down and then restarted. Perform the following steps:

Make sure that the job scheduler is set to collect statistics every 10 minutes.

Start the workload generator and note the time. If the workload generator is still running against the database, stop the generator and restart it.

Allow time for some statistics to be generated (at least 20 minutes). Shorter time periods make it more difficult to determine where problems exist.

After an adequate period of time, run the `spreport.sql` script . Choose a start and end time that falls between the period that the workload generator was running. Name the report in a manner associated with the scenario, for example, for scenario 1 use `reportscn1.txt`, for Scenario 5 use `reportscn5.txt`, and so on.

Look for the generated report in the directory from which SQL*Plus was executed.

When resizing memory components, do not consume more memory than is actually required in order to meet the requirements of the objects given. For example, there is little point in resizing the shared pool to 500 MB. The purpose of the workshop is to be as realistic as possible.

Workshop Scenario (continued)

The company concerned has at present two OLTP users and two DSS users. The system was set up by a trainee DBA, and though it works, the performance is very slow. You have been invited in to get the present system working for 20 users. The company is about to expand to 10 of each type of user (twenty users in all). At the same time the company is unwilling to spend extra on new hardware components.

Therefore**,** the management has imposed a limit of 20 MB for the entire SGA.

Workshop Scenario 1 (Shared Pool)

- **Collect a snapshot.**
- **Provide a workload by executing the workload generator.**
- **Collect a snapshot.**
- **Generate a report.**

Workshop Scenario 1

Waits recorded on the latch "Shared pool, library cache" could be indicative of a small shared pool. However, before rushing out and increasing the SHARED_POOL_SIZE, it would be advisable to determine why the pool is too small. Some reasons are listed below:

- Many SQL statements stored in the SQL area that are only executed 1, and they differ only in literals in the where clause. A case could be made here for using bind variables, or setting CURSOR_SHARING. In order to determine these SQL statements examine the report created by STATSPACK, or query v$sql using the like option of the where clause to collect information regarding similar SQL statements.
- Examine what packages are loaded using the query shown below:

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

Workshop Scenario 1 (continued)

If these packages are used frequently, pin them in memory using the `Dbms_shared_pool.keep('Package_name')`; package. Another area to examine would be reducing the size of the package. Determine if there are large portions of the package that are not commonly used. If possible, separate procedures into packages that will have their contents utilized.

Examine SQL statements that are executed often using the following statement:

```
select sum(sharable_mem)
from V$SQLAREA where executions > 5;
```

With this information determine if the SQL statement can be converted into a procedure and stored as a package; this can assist users in sharing the same cursor.

After you have reduced the number of SQL statements as much as possible, run the following query:

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

Increase the shared pool in order to reduce *cache misses.* Record the increase received for each increase in the shared pool in order to determine if the extra memory is worth the increase received.

Data Dictionary Cache

The data dictionary cache cannot be independently resized. The Oracle server automatically assigns space from the `Shared_Pool_Size` parameter for the shared SQL area, and the data dictionary cache. Most of the increase to `SHARED_POOL_SIZE` is a located to the shared SQL area.

In order to determine the hit ratio of the data dictionary cache, run the query:

```
select parameter, gets, getmisses
from v$rowcache;
```

The result of this query is a row for each segment of the data dictionary cache. Each area then can be checked for usage. For example, if there is a large number of gets on `dc_sequences`, this is probably due to sequence numbers not being cached. To reduce the number of gets on `dc_sequences`, examine increasing the number of sequence numbers cached.

Workload Scenario 2 (Buffer Cache)

- **Collect a snapshot.**
- **Provide a workload by executing the workload generator.**
- **Collect a snapshot.**
- **Generate a report.**

ORACLE

Workshop Scenario 2

The first indication that the buffer cache is too small is waits on free buffer waits event. The cache buffers LRU chain latch might also indicate that the buffer cache is too small. Waits on the latch may also signify that the DBWR process is not able to keep up with the work load.

In order to determine which problem is causing the latch contention, examine the number of writes in the file statistics found in the STATSPACK report.

On the front page of the STATSPACK report, the section named "Instance Efficiency Percentages" lists the important ratios of the instance. For this scenario, the values of `"Buffer Hit%:"` are of interest.

Values for the `"Buffer Hit%:"` depend on individual systems. Ideally, this value should be close to 100 percent; however, there might be several reasons why this goal cannot be realized. A low percentage indicates that there are a lot of buffers being read into the database buffer cache.

Before increasing the size of the database buffer cache, you should examine what SQL statements are being run against the database. You are looking for statements that cause a high number of *buffer gets* and how many times these statements are executed. If a statement is executed only a few times, it may not be worth the effort of tuning. However, a statement that is executed many times, and has a high number of *buffer gets* is an ideal candidate for SQL tuning.

Workshop Scenario 2 (continued)

STATSPACK lists the SQL statements that have been executed on the database. The first such listing orders the statements according to the number of gets. Examine the top statements, keeping in mind that packages are also listed here, not just the SQL statements. When a candidate statement is found, examine the SQL statement to determine if:

- There are indexes that could be created to assist in using less blocks
- There is a better way to write the statement in order to return the same data, but use less blocks
- Investigate if the statement has to be executed so often. Can output be generated and then shared between users?

After you have examined the SQL statements, and exhausted all means to reduce the number of buffers, then consider changing the size of the buffer cache. You must determine two items before changing the buffer cache size:

1. What is the current size?

   Query the data dictionary in order to determine the current size of the buffer cache. This can be performed by:

   ```
   Show parameter
   Select * from v$sgastat where name ='db_block_buffers';
   ```

   Front page of the STATSPACK report

2. What is an appropriate size for the buffer cache?

   Determine the new value of the buffer cache by using the `db_cache_advice` parameter. This parameter can have one of three values: `OFF, ON` and `READY`.

Setting the value to ON will start collecting the required statistics. The value can be changed by either:

- Editing the `init.ora` file and bouncing the database
- Using the `alter system set db_cache_advice = on;` command

When values has been set to `ON`, let the system run the required scripts in order to collect information regarding buffer usage.

After the database executes a typical load, query the `$db_cache_advice` view and set a new value for the `db_cache_size` parameter. When a new size has been determined, use the following command to dynamically change the size of the cache.

```
alter system set db_block_buffers = new_value;
```

Run a test load with this new value. Collect the statistics again. If the increase has resolved the problem then change the value in the `init.ora` file.

Workshop Scenario 3 (Redo Log Buffer)

- **Collect a snapshot.**
- **Provide a workload by executing the workload generator.**
- **Collect a snapshot.**
- **Generate a report.**

Workshop Scenario 3

Waits on the event *log buffer space* is an indication that your log buffer is too small.

On the first page of the STATSPACK report there is section named "Instance Efficiency Percentages." Note the value of the statistic `redo no wait`. While this statistic's ideal value of 100 percent is seldom achieved, any lesser value indicates that the Redo Log Buffer is not correctly sized. Consider reducing the amount of redo created by the use of `no logging` in appropriate statements.

Query the data dictionary in order to determine the current size of the Redo Log Buffer.

In order to determine an estimate on the increase required, examine the amount of redo that is generated; this is found on the first page of the STATSPACK report, under the heading "Load Profile."

Edit the `init.ora` file to set a new size for the redo log buffer.

Bounce the database.

Workshop Scenario 3 (continued)

Rerun the workload generator and collect the statistics again. If the increase has resolved the problem; and then, if the change was vast, repeat the process with a larger redo log buffer.

Note: The redo log buffer does not always have the size stipulated in the parameter file. This is due to minimum size, and rounding upwards to the nearest Oracle block. To confirm the actual redo log buffer size use the `v$sgastat` view.

Workshop Scenario 4 (Indexes)

- **All the indexes have been eliminated.**
  - **Result full table scans instead of using indexes**
- **Run workload generator**
- **Examine the SQL statements executed**
- **Determine which indexes should be recreated**

Workshop Scenario 4

Several indexes have been deleted and performance has decreased. The results are seen in the STATSPACK report where there are many indications that *untuned* SQL is running. For example:

- The buffer cache hit ratio is lower.This can be seen on the first page of the STATSPACK report in the Load Profile section.
- There are more waits on the free buffer waits event.
- There are more full table scans occurring.

This indicates that, because of incorrectly written SQL scripts, or missing or incorrect indexes, the database is performing too many full table scans.

In order to resolve the problem, you must determine which SQL statements are run on the database during a normal workload. Either use the SQL Trace utility or examine the top resource users in the STATSPACK report to collect a representation of these SQL statements.

After the appropriate statements are collected, examine the WHERE clauses. Any columns referenced in the WHERE clause are good candidates for indexes.

Workshop Scenario 4 (continued)

In order to determine if an index would be used by the optimizer, you must look at the expected number of rows to be returned. The more rows to be returned, the less likely an index will improve performance.

Confirm that the required indexes are present and enabled. The index might have been disabled for some reason. If the index is not present, then create the index.

Workshop Scenario 5 (Rollback Segments)

- **Create a workload by running:**
- **A daytime workload with the workload generator**
- **A nighttime workload with the `wk3.sh` script**

You may not use more disk space than is already used.

ORACLE

Workshop Scenario 5

The company has 20 users that log on to the database during daytime hours and perform OLTP transactions. To simulate this workload use the workload generator and set both order entry and shipping to `OLTP1`. During the nighttime, there are five users that log on and perform refreshes of certain tables. Currently, the system is set up with one very large rollback segment.

For the STATSPACK report collected between normal work hours (that is, when the workload generator is running), there is a section named "Buffer Wait Statistics." In this section should be found a statistic "undo header" which indicates that there is contention for the rollback segment header blocks.

In resolving this problem, do not forget that the nighttime run has to use the large rollback segment.

Use the two methods to resolve this problem.

Workshop Scenario 5 (continued)

Pre-Oracle9*i* Method

In order to resolve contention on the rollback segment header, more rollback segments have to be created. In order to do this without using more disk space requires some planning. For the nighttime run there should be few very large rollback segments, and smaller rollback segments during the daytime run. The scenario is further complicated because the nighttime run must not have access to the smaller rollback segments in case Oracle server allocates a small rollback segment to a large transaction, in which case the large transaction is likely to fail.

Oracle9*i* Method

In Oracle9*i,* the problems around the number and size of rollback segments has been eliminated. Instead of creating rollback segments, the DBA only has to create an undo tablespace that is large enough.

Workload Scenario 6 (Sorting)

- **Create a data warehouse type workload by running the workload generator. Have both sets of users using `query1`.**
- **Make sure that STATSPACK is collecting statistics.**
- **After statistics have been collected, generate a report.**
- **Examine the report.**

ORACLE

Workshop Scenario 6

During the evening hours, the company has twenty batch programs that log on and create a series of reports for management. This requires a lot of sorting. Currently, the scripts are completing. However, management would appreciate a more rapid completion.

The first step is to run the scripts and collect the STATSPACK report. Also make a note of how many transactions are completed during your running period (should be at least ten minutes).

Because we are concerned about sorts, the tendency is to jump straight to the Instance Activity stats and look for the values of Sorts (Disk), Sorts (Memory), and Sorts (Rows). However, doing so ignores some good information found on the front page.

On the front page, look at the buffer hit ratio. In a data warehouse environment we would expect this ratio to drop, however, combining this information with the high number of Buffer Busy Waits indicates that the buffer cache is too small for the number of sorts taking place. So you would likely wish to increase the size of the buffer cache.

Moving to the Instance Activity report, you find that a large number of sorts are having to go to disk. Ideally, no sorts should go to disk; however, accomplishing this has a high cost in memory. The ratio of sorts (Disk) to sorts (Memory) should be less than 5 percent.

Workshop Scenario 6 (continued)

Increasing the value of SORT_AREA_SIZE will resolve this issue, and more sorts will be performed in memory. The disadvantage of increasing the SORT_AREA_SIZE parameter is that more memory is consumed, and after completing the sort run, this memory is not released back to the operation system. It is released, but only to the sessions UGA. Therefore many users performing large sorts will consume memory.

Try increasing the value of the SORT_AREA_SIZE parameter by a factor of 10, and see what the effect is on the number of sorts (Disk).

In releases prior to Oracle9*i,* there was nothing a DBA could do in order to resolve this problem. A choice had to be made between increasing memory consumption (by changing the value of SORT_AREA_SIZE), or live with sorts going to disk.

In Oracle9*i* it is possible to change the manner in which sorts are handled in memory. Instead of allocating memory in terms of SORT_AREA_SIZE (which would then be locked onto one session), Oracle9*i* allows one memory allocation to be used for all sessions, thereby enabling the sharing of memory after the sort has completed.

The new parameter is PGA_AGGREGATE_TARGET and ranges in size from 10 MB to 4000 GB. When setting the value for this parameter, keep in mind that this is the total sort area for all sessions performing sorts.

In order to enable use of PGA_AGGREGATE_TARGET, the WORKAREA_SIZE_POLICY to AUTO parameter must be set.

Workload Scenario 7
(Reading STATSPACK Report)

- **This is an actual STATSPACK report.**
- **Examine the report in Appendix C.**
- **Make a list of problems and solutions.**

Workshop Scenario 7

The report in Appendix C is an actual production STATSPACK report. You are going to use this report to determine what should be looked at on the system in order to increase performance. For this exercise there will not be any actions processed on the database.

When reading the STATSPACK report and deciding upon an action, you must consider all the information given. Never just look at the front page, and discard all the rest. Among the many items on the front page of the STATSPACK that you should take note of are:

1. The length of the period over which the report is taken. This figure gets used when examining other statistics, for example, when looking at how many times a SQL statement was run during the collection period.
   In the example, the period of collection is 239.93 minutes (roughly 4 hours).
2. Examine the load profile statistics. It would be useful to compare this report with a previous report (a benchmark). Doing so could provide important information on how the load has changed, which would then point the way to where a bottleneck is occurring.
3. Take note of the percentage for "Rollback per transaction." A value of 12.43% indicates that roughly 1 transaction in 8 gets rolled back. Rolling back increases the workload of the database.

Workshop Scenario 7 (continued)

3. Examine why there is such a high amount of rollback occurring. Check that users have received the correct training on the system that they are using. Confirm that the code itself is not making use of unnecessary rollback. Often, educating the users and developers on the effects of rollback drastically reduces the number of rollbacks performed. Reducing the amount of rollbacks performed will also reduce the amount of redo generated.
4. Examine the "Instance Efficiency Percentages." The target is to have all of these figures at 100 percent; however, it is uncommon to achieve this target for a production database. Again it would be very useful to compare these figures against the benchmark report. Statistics that have altered dramatically could provide an indication of how the system has changed, and therefore where the bottlenecks might be occurring.
5. The last area to examine on the front page is the "Top 5 Wait Events." The report is a summary of the wait events view from the next page. The purpose of highlighting the top five wait events is to reinforce the importance of resolving the areas of high total wait time. Both the front page Top 5 report, and the complete wait event report on the following page, sort the background process wait events, such as `pmon` timer, to the bottom of the list.
6. Examining the report will show that there is a total wait time of 1439745.1 seconds for the enqueue event. This should be investigated further by examining the `v$enqueue_stat` view. From this view you can determine where the bottleneck is occurring, and thereby attempt to eliminate the waits. For example, if there are waits on the 'ST' enqueue (which is the space management enqueue), the likely cause is dynamic extent management, and you need to investigate using locally managed tablespaces, or pre-allocate the extents.
   After examining the first page of the report, the next section of interest contains the SQL statements executed on the database. The first of these reports sorts the SQL statements in accordance with buffer gets. The first statement listed is the statement consuming the most buffer gets.
7. In the example, the top statement has 174,229,988 buffer gets during the four hour monitoring period. The statement is executed 12,762 times, meaning that there are 13,652 buffer gets per execution. This presents a good candidate for some tuning. Examine the following:
   a. Do indexes exist on the appropriate columns?
   b. Are the indexes valid?
   c. Are the indexes being used? (Check the explain plan for this statement.)

   Due to the number of executions, and the high number of buffer gets that this statement generates, it is highly likely to give the biggest gain for time spent in tuning.
8. Another part of the SQL statement report to examine are the SQL statements sorted by physical reads. The first few statements are the ones that generate the best time returned on tuning.

Workshop Scenario 7 (continued)

9. Another good use for the SQL statement section of the STATSPACK report is to examine what statements are being run on the database. In order to increase the benefits of searching through the statements, a DBA should understand what the application does, and why. However, even without that level of understanding, some statements will stand out.
   For example, in our report in the section that sorts the SQL statements by executions is found the statements:
   - `SELECT SYSDATE FROM SYS.DUAL` which is executed 652,994 times.
   - `SELECT SYSDATE FROM DUAL` which is executed 532,476 times.

   The first item of concern is why are two cursors being used, instead of sharing cursors by making both statements the same.
   The second item of concern is why, in a 4-hour period (or 240 minutes, or 14,400 seconds), there is a total of 1,185,470 queries regarding the system date. This works out to be roughly 82 queries per second and would warrant an investigation as to why these are occurring. However, because neither of these statements are featured on the other SQL statement lists, the performance impact is minor and not worthy of a prolonged investigation.

D

Example of STATSPACK Report

```
STATSPACK report for
DB Name         DB Id    Instance     Inst Num Release     OPS Host
------------ ----------- ------------ -------- ----------- --- ------------
dba01        123456789   dba01               # 8.1.7.0.0   NO  dba01
                Snap Id     Snap Time      Sessions
                ------- ------------------ --------
 Begin Snap:          2 02-May-01 12:00:10    2,519
   End Snap:          4 02-May-01 16:00:06    2,519
    Elapsed:                239.93 (mins)
Cache Sizes
~~~~~~~~~~~
           db_block_buffers:      90000          log_buffer:     524288
              db_block_size:       8192    shared_pool_size:  262144000
Load Profile
~~~~~~~~~~~~                              Per Second       Per Transaction
                                    ---------------       ---------------
                  Redo size:            132,801.80             11,445.59
              Logical reads:             43,217.78              3,724.75
              Block changes:                710.62                 61.25
             Physical reads:              2,649.21                228.32
            Physical writes:                181.81                 15.67
                 User calls:              1,545.14                133.17
                     Parses:                630.82                 54.37
                Hard parses:                  1.84                  0.16
                      Sorts:                289.81                 24.98
                     Logons:                  0.63                  0.05
                   Executes:                752.20                 64.83
               Transactions:                 11.60
  % Blocks changed per Read:    1.64    Recursive Call %:   11.57
 Rollback per transaction %:   12.43       Rows per Sort:    9.08
Instance Efficiency Percentages (Target 100%)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
            Buffer Nowait %:   99.97       Redo NoWait %:  100.00
            Buffer  Hit   %:   93.87    In-memory Sort %:  100.00
            Library Hit   %:   99.79        Soft Parse %:   99.71
         Execute to Parse %:   16.14         Latch Hit %:   97.52
Parse CPU to Parse Elapsd %:   71.99     % Non-Parse CPU:  100.00
 Shared Pool Statistics        Begin   End
                               ------  ------
             Memory Usage %:   89.60   89.68
    % SQL with executions>1:   59.55   78.54
  % Memory for SQL w/exec>1:   44.22   65.22
Top 5 Wait Events
~~~~~~~~~~~~~~~~~                                                    Wait     % Total
Event                                               Waits  Time (cs)   Wt Time
-------------------------------------------- ------------ ------------ -------
enqueue                                           472,825  143,974,510   82.43
db file sequential read                        15,863,265   15,686,892    8.98
PX Deq: Execution Msg                              33,383    5,719,203    3.27
latch free                                      1,618,619    2,300,843    1.32
db file scattered read                            717,629    1,364,228     .78
          -------------------------------------------------------------
```

Wait Events for DB: dba01 Instance: dba01 Snaps: 2 -4

-> cs - centisecond - 100th of a second
-> ms - millisecond - 1000th of a second
-> ordered by wait time desc, waits desc (idle events last)

Event	Waits	Timeouts	Total Wait Time (cs)	Avg wait (ms)	Waits /txn
enqueue	472,825	471,414	##########	3045	2.8
db file sequential read	15,863,265	0	15,686,892	10	95.0
PX Deq: Execution Msg	33,383	27,499	5,719,203	1713	0.2
latch free	1,618,619	1,515,932	2,300,843	14	9.7
db file scattered read	717,629	0	1,364,228	19	4.3
PX Deq: Table Q Sample	5,406	5,129	1,040,147	1924	0.0
Replication Dequeue	92,127	2,617	929,538	101	0.6
PX Deq Credit: send blkd	4,556	4,156	857,664	1882	0.0
PX Deq: Execute Reply	5,636	1,474	535,227	950	0.0
PX Deque wait	91,970	2,211	468,056	51	0.6
global cache lock open x	101	78	453,806	44931	0.0
PX Deq: Table Q Get Keys	2,196	2,089	436,612	1988	0.0
PX Deq: Table Q Normal	4,042	1,215	354,350	877	0.0
buffer busy waits	160,335	978	286,225	18	1.0
log file parallel write	319,798	0	88,161	3	1.9
file open	78,950	0	58,851	7	0.5
log file sync	107,835	109	58,767	5	0.6
SQL*Net more data to client	2,331,201	0	22,557	0	14.0
PX qref latch	151	129	13,621	902	0.0
SQL*Net message from dblink	19,350	0	8,834	5	0.1
log file sequential read	32,674	0	3,435	1	0.2
db file parallel read	369	0	2,842	77	0.0
control file parallel write	5,073	0	2,814	6	0.0
control file sequential read	9,687	0	665	1	0.1
log file switch completion	76	0	573	76	0.0
process startup	86	0	559	65	0.0
db file parallel write	368,945	0	507	0	2.2
library cache pin	271	0	501	18	0.0
PX Deq: Signal ACK	178	36	328	18	0.0
LGWR wait for redo copy	5,140	65	315	1	0.0
PX Deq: Parse Reply	389	0	191	5	0.0
refresh controlfile command	801	0	173	2	0.0
PX Deq: Join ACK	436	0	156	4	0.0
local write wait	24	0	140	58	0.0
SQL*Net break/reset to clien	3,539	0	102	0	0.0
log file single write	34	0	29	9	0.0
SQL*Net message to dblink	19,355	0	19	0	0.1
file identify	153	0	19	1	0.0
PX Deq: Msg Fragment	298	0	18	1	0.0
direct path read	3,110	0	5	0	0.0
direct path write	1,049	0	4	0	0.0
PX Deq: Table Q qref	74	0	2	0	0.0

Event	Waits	Timeouts	Total Wait Time (cs)	Avg wait (ms)	Waits /txn
SQL*Net break/reset to dblin	10	0	1	1	0.0
single-task message	1	0	1	10	0.0
PX Deq Credit: need buffer	19	0	0	0	0.0
buffer deadlock	6	6	0	0	0.0
SQL*Net message from client	22,245,825	0	###########	1478	133.2
PX Idle Wait	31,957	31,475	6,474,543	2026	0.2
SQL*Net more data from clien	391,640	0	1,910,922	49	2.3
SQL*Net message to client	22,245,863	0	19,322	0	133.2

Background Wait Events for DB: dba01 Instance: dba01 Snaps: 2 -4
-> ordered by wait time desc, waits desc (idle events last)

Event	Waits	Timeouts	Total Wait Time (cs)	Avg wait (ms)	Waits /txn
log file parallel write	319,796	0	88,155	3	1.9
latch free	10,909	10,905	17,404	16	0.1
log file sequential read	32,674	0	3,435	1	0.2
control file parallel write	5,017	0	2,784	6	0.0
db file sequential read	1,464	0	1,106	8	0.0
file open	4,422	0	922	2	0.0
db file scattered read	333	0	513	15	0.0
db file parallel write	368,945	0	507	0	1.2
LGWR wait for redo copy	5,140	65	315	1	0.0
enqueue	3	0	258	860	0.0
control file sequential read	3,336	0	187	1	0.0
log file single write	34	0	20	9	0.0
file identify	85	0	17	2	0.0
direct path write	969	0	3	0	0.0
direct path read	1,258	0	2	0	0.0
rdbms ipc message	717,126	38,582	15,344,729	214	4.3
pmon timer	4,709	4,646	1,437,589	3053	0.0
smon timer	47	46	1,414,668	######	0.0

```
SQL ordered by Gets for DB: dba01  Instance: dba01  Snaps: 2 -4
-> End Buffer Gets Threshold:   10000
-> Note that resources reported for PL/SQL includes the resources used by
   all SQL statements called within the PL/SQL code.  As individual SQL
   statements are also reported, it is possible and valid for the summed
   total % to exceed 100

  Buffer Gets    Executions  Gets per Exec  % Total  Hash Value
--------------- ------------ -------------- ------- ------------
    174,229,988       12,762       13,652.2    28.0   4180317975
  SELECT "EXT_ACCOUNT_LINK"."COMPASS_ID",             "EXT_ACCO
UNT_LINK"."EXTACCT_IDENTIFIER",             "EXT_ACCOUNT_LINK".
"EXTSYS_ID"       FROM "EXT_ACCOUNT_LINK"      WHERE ( "EXT_AC
COUNT_LINK"."COMPASS_ID" = :as_compass_id ) AND            ( "E
XT_ACCOUNT_LINK"."EXTSYS_ID" = :as_extsys_id ) AND            (
     21,475,679        1,253       17,139.4     3.5   3671942761
select  /*+  INDEX(CA compass_account_ndx6)  +*/ ROWIDTOCHAR(CA.
rowid ) ,ML.MARKET_ID ,NVL(RTRIM(AD.ADDRESS_APT_DESIGNATOR),' ')
 ,NVL(RTRIM(AD.ADDRESS_APT_NUM),' ') ,NVL(RTRIM(AD.ADDRESS_ATTEN
TION),' ') ,NVL(RTRIM(AD.ADDRESS_CITY),' ') ,NVL(RTRIM(AD.ADDRES
S_COUNTRY),' ') ,NVL(RTRIM(AD.ADDRESS_FORMAT),' ') ,NVL(RTRIM(AD
     17,919,523          365       49,094.6     2.9    405379982
( SELECT "INV_MOVEMENT_SUBLN_A"."MVMNT_LOCATION_ID", "INV_MOVEME
NT_SUBLN_A"."ACTIVITY_TYPE", "INV_MOVEMENT_SUBLN_A"."MVMNT_OID",
 "INV_MOVEMENT_SUBLN_A"."MVMNT_LINE_SEQ", "INV_MOVEMENT_SUBLN_A"
."MVMNT_SUBLINE_SEQ", "INV_MOVEMENT_SUBLN_A"."SYS_CREATION_DATE"
, "INV_MOVEMENT_SUBLN_A"."SYS_UPDATE_DATE", "INV_MOVEMENT_SUBLN_
     12,562,692    3,703,535            3.4     2.0   3533983708
  SELECT  "SECURITY_INFO"."WINDOW" ,           "SECURITY_INFO"."
CONTROL" ,           "SECURITY_INFO"."STATUS" ,           "SECUR
ITY_USERS"."PRIORITY"     FROM "SECURITY_INFO" ,           "SECU
RITY_USERS"     WHERE ( "SECURITY_USERS"."NAME" = "SECURITY_INFO
"."USER_NAME" ) and          ( "SECURITY_INFO"."WINDOW" = :winnv
     12,469,136            4    3,117,284.0     2.0   3966365613
  SELECT Distinct("INV_PURCHASE_ORDER"."PURO_OID"),           "INV_PUR
CHASE_ORDER"."STATUS",              "INV_PURCHASE_ORDER"."SYS_CR
EATION_DATE",              "INV_PURCHASE_ORDER"."OPERATOR_ID",
            "INV_PURCHASE_ORDER"."LOCATION_ID",             "IN
V_PURCHASE_ORDER"."PURO_NEEDBY_DATE",           "INV_PURCHASE
     11,632,584       10,460        1,112.1     1.9   3903562577
  SELECT  "SALES_REPRESENTATIVE"."SALESREP_ID" ,           "SALE
S_REPRESENTATIVE"."SALESREP_NAME"     FROM "LOCATION_SALESREP_LI
NK" ,           "SALES_REPRESENTATIVE"     WHERE ( "LOCATION_SAL
ESREP_LINK"."SALESREP_ID" = "SALES_REPRESENTATIVE"."SALESREP_ID"
 ) and          ( "LOCATION_SALESREP_LINK"."LOCATION_ID" = :as_l
```

```
SQL ordered by Gets for DB: dba01  Instance: dba01  Snaps: 2 -4
-> End Buffer Gets Threshold:   10000
-> Note that resources reported for PL/SQL includes the resources used by
   all SQL statements called within the PL/SQL code.  As individual SQL
   statements are also reported, it is possible and valid for the summed
   total % to exceed 100
  Buffer Gets    Executions  Gets per Exec  % Total  Hash Value
--------------- ------------ -------------- ------- ------------
6,235,002            2    3,117,501.0     1.0   1491625703
  SELECT Distinct("INV_PURCHASE_ORDER"."PURO_OID"),     "INV_PUR
CHASE_ORDER"."STATUS",              "INV_PURCHASE_ORDER"."SYS_CR
EATION_DATE",              "INV_PURCHASE_ORDER"."OPERATOR_ID",
            "INV_PURCHASE_ORDER"."LOCATION_ID",              "IN
V_PURCHASE_ORDER"."PURO_NEEDBY_DATE",              "INV_PURCHASE
      5,768,648       47,284          122.0     0.9   3627000592
  SELECT  "MARKET"."MARKET_ID" ,           "MARKET"."MARKET_NAME
"     FROM "MARKET" ,           "MARKET_POLICY"     WHERE ( "MAR
KET"."MARKET_ID" = "MARKET_POLICY"."MARKET_ID" ) and          (
( "MARKET"."EXTSYS_ID" = DECODE( :as_extsys_id, '*', market.exts
ys_id, :as_extsys_id ) ) and          ( "MARKET_POLICY"."MKTPOL_
      5,432,674        1,278        4,250.9     0.9   1349835727
SELECT distinct  "SECURITY_USERS"."NAME", "SECURITY_USERS"."DESC
RIPTION"   FROM "LOCATION",  "SECURITY_USERS",  "WORKSTATION
_SETTINGS",  "USER_MARKET_LINK"   WHERE   ( "WORKSTATION_SETT
INGS"."WS_HOSTNAME" = :as_hostname )    AND ( "WORKSTATION_SETT
INGS"."LOCATION_ID" = "LOCATION"."LOCATION_ID" )    AND ( "LOCA
      4,069,186       13,253          307.0     0.7   2947143525
  SELECT  "ITEM_DEFINITION"."ITEM_ID" ,           "ITEM_DEFINITI
ON"."SYS_CREATION_DATE" ,           "ITEM_DEFINITION"."SYS_UPDAT
E_DATE" ,           "ITEM_DEFINITION"."OPERATOR_ID" ,
"ITEM_DEFINITION"."APPLICATION_ID" ,           "ITEM_DEFINITION"
."DL_SERVICE_CODE" ,           "ITEM_DEFINITION"."DL_UPDATE_STAM
      4,056,847           23      176,384.7     0.7   2734622026
DECLARE job BINARY_INTEGER := :job; next_date DATE := :mydate;
broken BOOLEAN := FALSE; BEGIN declare rc binary_integer; begin
rc := sys.dbms_defer_sys.push(destination=>'dba01.WORLD', stop
_on_error=>TRUE, execution_seconds=>600, delay_seconds=>180, par
allelism=>1); end; :mydate := next_date; IF broken THEN :b := 1;
      3,873,264           28      138,330.9     0.6   3293673928
SELECT DISTINCT "VENDOR"."VENDOR_ID", "VENDOR"."VENDOR_NAME", "V
ENDOR"."VENDOR_SDESC", "VENDOR"."VENDOR_TEL_NO", "VENDOR"."VENDO
R_FAX_NO", "VENDOR"."VENDOR_CONTACT_NAME" FROM "VENDOR", "VENDOR
_ITEM" WHERE ( "VENDOR"."VENDOR_ID" = "VENDOR_ITEM"."VENDOR_ID"
) ORDER BY "VENDOR"."VENDOR_ID" ASC, "VENDOR"."VENDOR_NAME" ASC
      3,845,686       45,430            8.5     0.6   3709651884
insert into SYS.DEF$_AQCALL  (q_name, msgid, corrid, priority
, state, delay, expiration,   time_manager_info, local_order_no,
 chain_no, enq_time, step_no, enq_uid,   enq_tid, retry_count, e
xception_qschema, exception_queue, recipient_key,   dequeue_msgi

          -------------------------------------------------------------
```

```
SQL ordered by Reads for DB: dba01  Instance: dba01  Snaps: 2 -4
-> End Disk Reads Threshold:    1000
 Physical Reads  Executions  Reads per Exec % Total  Hash Value
--------------- ------------ -------------- ------- ------------
      8,087,240        1,253        6,454.3    21.2   3671942761
select  /*+  INDEX(CA compass_account_ndx6)  +*/ ROWIDTOCHAR(CA.
rowid ) ,ML.MARKET_ID ,NVL(RTRIM(AD.ADDRESS_APT_DESIGNATOR),' ')
 ,NVL(RTRIM(AD.ADDRESS_APT_NUM),' ') ,NVL(RTRIM(AD.ADDRESS_ATTEN
TION),' ') ,NVL(RTRIM(AD.ADDRESS_CITY),' ') ,NVL(RTRIM(AD.ADDRES
S_COUNTRY),' ') ,NVL(RTRIM(AD.ADDRESS_FORMAT),' ') ,NVL(RTRIM(AD
      1,171,800          241        4,862.2     3.1   2416933180
select ROWIDTOCHAR(CA.rowid ) ,ML.MARKET_ID ,NVL(RTRIM(AD.ADDRES
S_APT_DESIGNATOR),' ') ,NVL(RTRIM(AD.ADDRESS_APT_NUM),' ') ,NVL(
RTRIM(AD.ADDRESS_ATTENTION),' ') ,NVL(RTRIM(AD.ADDRESS_CITY),' '
) ,NVL(RTRIM(AD.ADDRESS_COUNTRY),' ') ,NVL(RTRIM(AD.ADDRESS_FORM
AT),' ') ,NVL(RTRIM(AD.ADDRESS_PRIMARY_LN),' ') ,NVL(RTRIM(AD.AD
        797,761          364        2,191.7     2.1   2188104695
  SELECT "CAS_RESULT"."CAS_COMPASS_ID"      FROM "CAS_RESULT"
    WHERE "CAS_RESULT"."CAS_COMPASS_ID" = :as_CompassID
        719,730           36       19,992.5     1.9    596848251
  SELECT "INV_MOVEMENT"."MVMNT_OID",     "INV_MOVEMENT"."MVMNT_S
TATUS",              "INV_MOVEMENT"."MVMNT_LOCATION_ID" || '-' |
|"INV_MOVEMENT"."ACTIVITY_TYPE"|| '-' || lpad( "INV_MOVEMENT"."M
VMNT_OID" ,6 , '0') trans_id,              "INV_MOVEMENT"."TRANS
_OUT_STS",              "INV_MOVEMENT"."SYS_CREATION_DATE",
        458,074        3,445          133.0     1.2   4195953210
  SELECT "POS_INVOICE_LINE"."LOCATION_ID",             "POS_INV
OICE_LINE"."POSINV_ACTIVITY",             "POS_INVOICE_LINE"."P
OSINV_ID",             "POS_INVOICE_LINE"."ACTIVITY_ID",
       "POS_INVOICE_LINE"."POSINVLN_ID",             "POS_INVOI
CE_LINE"."SYS_CREATION_DATE",             "POS_INVOICE_LINE"."S
        435,057          206        2,111.9     1.1   1396905682
SELECT TO_CHAR(max(CAS_CREDIT_DATE), 'MM/DD/YYYY HH24:MI:SS')
 FROM CAS_RESULT  WHERE CAS_COMPASS_ID = :as_CompassID
        432,358          206        2,098.8     1.1   464956795
  SELECT "CAS_RESULT"."CAS_ORDER_NUM",             "CAS_RESULT"
."SYS_CREATION_DATE",             "CAS_RESULT"."SYS_UPDATE_DATE
",             "CAS_RESULT"."OPERATOR_ID",             "CAS_RE
SULT"."APPLICATION_ID",             "CAS_RESULT"."DL_SERVICE_CO
DE",             "CAS_RESULT"."DL_UPDATE_STAMP",             "
        348,312            1        348,312.0     0.9   3411194506
  SELECT DISTINCT "POS_TRANSACTION"."LOCATION_ID",
"POS_TRANSACTION"."POSTRANS_PAYMENT_TYPE",             "POS_TRA
NSACTION"."POSTRANS_ID",             "POS_TRANSACTION"."COMPASS
_ID",             "POS_TRANSACTION"."POSTRANS_EXTACCT_NUM",
         "POS_TRANSACTION"."POSTRANS_DATE",             "POS_T
        333,311           22       15,150.5     0.9   3863558257
  SELECT DISTINCT "POS_ORDER"."LOCATION_ID",             "POS_O
RDER"."ORDER_ACTIVITY",             "POS_ORDER"."ORDER_ID",
         "POS_ORDER"."COMPASS_ID",             "ORDER_FULFILLM
ENT"."ORDFILL_ID",             "ORDER_FULFILLMENT"."SYS_CREATIO
```

```
SQL ordered by Reads for DB: dba01  Instance: dba01  Snaps: 2 -4
-> End Disk Reads Threshold:    1000
 Physical Reads  Executions  Reads per Exec % Total  Hash Value
--------------- ------------ -------------- ------- ------------
      8,087,240        1,253        6,454.3    21.2   3671942761
select  /*+  INDEX(CA compass_account_ndx6)  +*/ ROWIDTOCHAR(CA.
rowid ) ,ML.MARKET_ID ,NVL(RTRIM(AD.ADDRESS_APT_DESIGNATOR),' ')
 ,NVL(RTRIM(AD.ADDRESS_APT_NUM),' ') ,NVL(RTRIM(AD.ADDRESS_ATTEN
TION),' ') ,NVL(RTRIM(AD.ADDRESS_CITY),' ') ,NVL(RTRIM(AD.ADDRES
S_COUNTRY),' ') ,NVL(RTRIM(AD.ADDRESS_FORMAT),' ') ,NVL(RTRIM(AD
      1,171,800          241        4,862.2     3.1   2416933180
select ROWIDTOCHAR(CA.rowid ) ,ML.MARKET_ID ,NVL(RTRIM(AD.ADDRES
S_APT_DESIGNATOR),' ') ,NVL(RTRIM(AD.ADDRESS_APT_NUM),' ') ,NVL(
RTRIM(AD.ADDRESS_ATTENTION),' ') ,NVL(RTRIM(AD.ADDRESS_CITY),' '
) ,NVL(RTRIM(AD.ADDRESS_COUNTRY),' ') ,NVL(RTRIM(AD.ADDRESS_FORM
AT),' ') ,NVL(RTRIM(AD.ADDRESS_PRIMARY_LN),' ') ,NVL(RTRIM(AD.AD
        797,761          364        2,191.7     2.1   2188104695
  SELECT "CAS_RESULT"."CAS_COMPASS_ID"      FROM "CAS_RESULT"
    WHERE "CAS_RESULT"."CAS_COMPASS_ID" = :as_CompassID
        719,730           36       19,992.5     1.9    596848251
  SELECT "INV_MOVEMENT"."MVMNT_OID",     "INV_MOVEMENT"."MVMNT_S
TATUS",              "INV_MOVEMENT"."MVMNT_LOCATION_ID" || '-' |
|"INV_MOVEMENT"."ACTIVITY_TYPE"|| '-' || lpad( "INV_MOVEMENT"."M
VMNT_OID" ,6 , '0') trans_id,              "INV_MOVEMENT"."TRANS
_OUT_STS",              "INV_MOVEMENT"."SYS_CREATION_DATE",
        458,074        3,445          133.0     1.2   4195953210
  SELECT "POS_INVOICE_LINE"."LOCATION_ID",             "POS_INV
OICE_LINE"."POSINV_ACTIVITY",             "POS_INVOICE_LINE"."P
OSINV_ID",             "POS_INVOICE_LINE"."ACTIVITY_ID",
       "POS_INVOICE_LINE"."POSINVLN_ID",             "POS_INVOI
CE_LINE"."SYS_CREATION_DATE",             "POS_INVOICE_LINE"."S
        435,057          206        2,111.9     1.1   1396905682
SELECT TO_CHAR(max(CAS_CREDIT_DATE), 'MM/DD/YYYY HH24:MI:SS')
 FROM CAS_RESULT  WHERE CAS_COMPASS_ID = :as_CompassID
        432,358          206        2,098.8     1.1   464356795
  SELECT "CAS_RESULT"."CAS_ORDER_NUM",             "CAS_RESULT"
."SYS_CREATION_DATE",             "CAS_RESULT"."SYS_UPDATE_DATE
",             "CAS_RESULT"."OPERATOR_ID",             "CAS_RE
SULT"."APPLICATION_ID",             "CAS_RESULT"."DL_SERVICE_CO
DE",             "CAS_RESULT"."DL_UPDATE_STAMP",             "
        348,312            1        348,312.0     0.9   3411194506
  SELECT DISTINCT "POS_TRANSACTION"."LOCATION_ID",
"POS_TRANSACTION"."POSTRANS_PAYMENT_TYPE",             "POS_TRA
NSACTION"."POSTRANS_ID",             "POS_TRANSACTION"."COMPASS
_ID",             "POS_TRANSACTION"."POSTRANS_EXTACCT_NUM",
         "POS_TRANSACTION"."POSTRANS_DATE",             "POS_T
333,311           22       15,150.5     0.9   3863558257
  SELECT DISTINCT "POS_ORDER"."LOCATION_ID",             "POS_O
RDER"."ORDER_ACTIVITY",             "POS_ORDER"."ORDER_ID",
        "POS_ORDER"."COMPASS_ID",             "ORDER_FULFILLM
ENT"."ORDFILL_ID",             "ORDER_FULFILLMENT"."SYS_CREATIO
```

```
SQL ordered by Reads for DB: dba01  Instance: dba01  Snaps: 2 -4
-> End Disk Reads Threshold:    1000

 Physical Reads  Executions  Reads per Exec % Total  Hash Value
--------------- ------------ -------------- ------- ------------
N_DATE",              "ORDER_FULFILLMENT"."ORDFILL_TRACKING",

        327,408          145        2,258.0     0.9   1606276826
  SELECT  "CAS_RESULT"."CAS_COMPASS_ID" ,           "CAS_RESULT"
."CAS_RESULT_TYPE"     FROM "CAS_RESULT"      WHERE ( :as_Compas
sID = "CAS_RESULT"."CAS_COMPASS_ID" )

        320,452        7,303           43.9     0.8   2440585122
  SELECT  "CHECK_INOUT_LINE"."LOCATION_ID" ,           "CHECK_IN
OUT_LINE"."CHKINOUT_ACTIVITY" ,           "CHECK_INOUT_LINE"."CH
KINOUT_ID" ,           "CHECK_INOUT_LINE"."CHKINOUTL_ID" ,
     "CHECK_INOUT_LINE"."SYS_CREATION_DATE" ,           "CHECK_I
NOUT_LINE"."SALESREP_ID" ,           "CHECK_INOUT_LINE"."ITEM_ID

        281,487          782          360.0     0.7   1321409306
  SELECT  "POS_ORDER_SUBLINE"."LOCATION_ID" ,           "POS_ORD
ER_SUBLINE"."ORDER_ACTIVITY" ,           "POS_ORDER_SUBLINE"."OR
DER_ID" ,           "POS_ORDER_SUBLINE"."ORDERLN_ID" ,
 "POS_ORDER_SUBLINE"."ORDERSLN_ID" ,           "POS_ORDER_SUBLIN
E"."SYS_CREATION_DATE" ,           "POS_ORDER_SUBLINE"."SERIAL_I

        280,075           18       15,559.7     0.7   3733389583
  SELECT  "ORDER_FULFILLMENT"."LOCATION_ID" ,           "ORDER_F
ULFILLMENT"."ORDFILL_ID" ,           "ORDER_FULFILLMENT"."ORDFIL
L_STATUS" ,           "ORDER_FULFILLMENT_SUBLINE"."SERIAL_ID" ,
         "ORDER_FULFILLMENT"."ORDFILL_LOCATIONID" ,           "
ORDER_FULFILLMENT"."ORDFILL_ORDERID" ,           "ORDER_FULFILLM

        259,743           11       23,613.0     0.7   2742209160
SELECT /*+ ORDERED NO_EXPAND USE_NL(A2) */ A1.C1 C0,NVL(A1.C5,'
') C1,NVL(A1.C2,' ') C2,NVL(A1.C3,' ') C3,NVL(A1.C4,' ') C4,NVL(
A1.C6,' ') C5,NVL(A1.C10,0) C6 FROM :Q2112000 A1,(SELECT /*+ NO_
EXPAND ROWID(A3) */ A3."COMPASS_ID" C0,A3."POSINV_STATUS" C1,A3.
"POSINV_REMAINING_AR_AMT" C2,A3."MARKET_ID" C3 FROM "DEVOWNER"."

        192,141       13,253           14.5     0.5   2947143525
  SELECT  "ITEM_DEFINITION"."ITEM_ID" ,           "ITEM_DEFINITI
ON"."SYS_CREATION_DATE" ,           "ITEM_DEFINITION"."SYS_UPDAT

          -------------------------------------------------------
```

```
SQL ordered by Executions for DB: dba01  Instance: dba01  Snaps: 2 -4
-> End Executions Threshold:     100

 Executions   Rows Processed    Rows per Exec   Hash Value
------------ --------------- --------------- ------------
   3,703,535          370,614             0.1   3533983708
  SELECT  "SECURITY_INFO"."WINDOW" ,           "SECURITY_INFO"."
CONTROL" ,           "SECURITY_INFO"."STATUS" ,           "SECUR
ITY_USERS"."PRIORITY"     FROM "SECURITY_INFO" ,           "SECU
RITY_USERS"     WHERE ( "SECURITY_USERS"."NAME" = "SECURITY_INFO
"."USER_NAME" ) and          ( "SECURITY_INFO"."WINDOW" = :winna
     873,612          869,514             1.0    203525044
SELECT "LOCATION"."MARKET_ID" FROM "LOCATION" WHERE "LOCATION"."
LOCATION_ID" =:1
     652,994          652,993             1.0   2182723903
SELECT SYSDATE   FROM SYS.DUAL
     536,404          536,360             1.0   3258376580
  SELECT  "MARKET"."MARKET_ID" ,           "MARKET"."SYS_CREATIO
N_DATE" ,           "MARKET"."SYS_UPDATE_DATE" ,           "MARK
ET"."OPERATOR_ID" ,           "MARKET"."APPLICATION_ID" ,
    "MARKET"."DL_SERVICE_CODE" ,           "MARKET"."DL_UPDATE_S
TAMP" ,           "MARKET"."COMPANY_ID" ,           "MARKET"."MA
     532,476          532,473             1.0   1645188330
SELECT SYSDATE FROM DUAL
     451,430          451,430             1.0   3709651884
insert into SYSTEM.DEF$_AQCALL  (q_name, msgid, corrid, priority
, state, delay, expiration,   time_manager_info, local_order_no,
 chain_no, enq_time, step_no, enq_uid,   enq_tid, retry_count, e
xception_qschema, exception_queue, recipient_key,   dequeue_msgi
d, user_data)  values (:1, :2, :3, :4, :5, :6, :7, :8, :9, :10,
     298,348           10,263             0.0   2207347125
  SELECT  "SECURITY_INFO"."WINDOW" ,           "SECURITY_INFO"."
CONTROL" ,           "SECURITY_INFO"."STATUS" ,           "SECU
ITY_USERS"."PRIORITY"     FROM "SECURITY_INFO" ,           "SECU
RITY_USERS"     WHERE ( "SECURITY_USERS"."NAME" = "SECURITY_INFO
"."USER_NAME" ) and          ( ( "SECURITY_INFO"."WINDOW" = :as_
     230,434           38,386             0.2   31[illegible]6978
  SELECT  "TAX_ASSOCIATIONS"."STATE_CODE" ,           "TAX_ASSOC
IATIONS"."MARKET_ID" ,           "TAX_ASSOCIATIONS"."LOCATION_ID
" ,           "TAX_ASSOCIATIONS"."FUNCTION_ID" ,           "TAX_
ASSOCIATIONS"."SYS_CREATION_DATE" ,           "TAX_ASSOCIATIONS"
."SYS_UPDATE_DATE" ,           "TAX_ASSOCIATIONS"."OPERATOR_ID"
      99,743           99,742             1.0   3174277601
  SELECT "ITEM_POLICY"."ITEM_ID",           "ITEM_POLICY"."EF
FECTIVE_DATE",           "ITEM_POLICY"."SYS_CREATION_DATE",
          "ITEM_POLICY"."SYS_UPDATE_DATE",           "ITEM_
POLICY"."OPERATOR_ID",           "ITEM_POLICY"."APPLICATION_I
D",           "ITEM_POLICY"."DL_SERVICE_CODE",           “
98,607           98,605             1.0   1661068441
  SELECT  "WHSE_ITEM"."ITEM_ID" ,           "WHSE_ITEM"."WHSE_ID
" ,           "WHSE_ITEM"."TRACKING_IND"     FROM "WHSE_ITEM"
```

```
SQL ordered by Executions for DB: dba01  Instance: dba01  Snaps: 2 -4
-> End Executions Threshold:     100

 Executions   Rows Processed    Rows per Exec   Hash Value
------------ --------------- --------------- ------------
   WHERE ( "WHSE_ITEM"."ITEM_ID" = :as_item ) and          ( "WH
SE_ITEM"."WHSE_ID" = :as_warehouse )
      98,215           96,727             1.0   3239320604
  SELECT  "ITEM_WAC_ACCUMULATOR"."WAC_LOCATION_ID" ,           "
ITEM_WAC_ACCUMULATOR"."ITEM_ID" ,           "ITEM_WAC_ACCUMULATO
R"."LAST_UPDATE_DATE" ,           "ITEM_WAC_ACCUMULATOR"."SYS_CR
EATION_DATE" ,           "ITEM_WAC_ACCUMULATOR"."SYS_UPDATE_DATE
" ,           "ITEM_WAC_ACCUMULATOR"."OPERATOR_ID" ,           "
      95,244           90,281             0.9   1418052366
  SELECT  "ITEM_DEFINITION"."ITEM_ID" ,           "ITEM_DEFINITI
ON"."SYS_CREATION_DATE" ,           "ITEM_DEFINITION"."SYS_UPDAT
E_DATE" ,           "ITEM_DEFINITION"."OPERATOR_ID" ,
"ITEM_DEFINITION"."APPLICATION_ID" ,           "ITEM_DEFINITION"
."DL_SERVICE_CODE" ,           "ITEM_DEFINITION"."DL_UPDATE_STAM
      87,594           87,594             1.0    254026706
update SYSTEM.DEF$_AQCALL  set dscn = :1, cscn = :2  where rowid
 = :3
      87,590          448,047             5.1    568505420
SELECT /*+ ORDERED USE_NL(P) */ aq.step_no, P.sname, P.oname, aq
.chain_no, aq.user_data FROM system.def$_aqcall aq, system.repca
t$_repprop P WHERE  aq.enq_tid = :tid AND P.recipient_key = aq.r
ecipient_key AND P.how = 1 AND P.dblink = :dest ORDER BY aq.enq_
tid, aq.step_no
      86,861           75,704             0.9   1084722568
  SELECT  "SERIAL_ITEM_INV"."SERIAL_NUMBER" ,           "SERIAL_
ITEM_INV"."APPLICATION_ID" ,           "SERIAL_ITEM_INV"."DL_SER
VICE_CODE" ,           "SERIAL_ITEM_INV"."DL_UPDATE_STAMP" ,
       "SERIAL_ITEM_INV"."ACT_ISSUE_DATE" ,           "SERIAL_IT
EM_INV"."ITEM_ID" ,           "SERIAL_ITEM_INV"."POOL" ,
      86,579                0             0.0    256517785
delete from system.def$_lob     where (enq_tid = :1)
      86,579                0             0.0    307735379
delete from system.def$_aqcall     where (enq_tid = :1)
      86,579                0             0.0   1565159147
delete from system.def$_calldest     where (enq_tid = :1)
      75,981           75,978             1.0   3356946589
  SELECT  "MARKET_POLICY"."MARKET_ID" ,           "MARKET_POLICY
          ------------------------------------------------------------
SQL ordered by Version Count for DB: dba01  Instance: dba01  Snaps: 2 -4
-> End Version Count Threshold:        20
 Version
   Count  Executions   Hash Value
------- ------------ ------------
      31        1,155   1668540021
select i.obj# from ind$ i, obj$ o    where i.obj# = o.obj# and i
.bo# = :1 and o.name = :2
          ------------------------------------------------------------
```

Instance Activity Stats for DB: dba01 Instance: dba01 Snaps: 2 -4

Statistic	Total	per Second	per Trans
CPU used by this session	93,508,307	6,495.4	559.8
CPU used when call started	5,274,242	366.4	31.6
CR blocks created	213,017	14.8	1.3
DBWR buffers scanned	12,552,443	871.9	75.2
DBWR checkpoint buffers written	2,260,416	157.0	13.5
DBWR checkpoints	50	0.0	0.0
DBWR free buffers found	12,423,954	863.0	74.4
DBWR lru scans	132,887	9.2	0.8
DBWR make free requests	146,146	10.2	0.9
DBWR revisited being-written buff	7	0.0	0.0
DBWR summed scan depth	12,552,443	871.9	75.2
DBWR transaction table writes	128,583	8.9	0.8
DBWR undo block writes	299,657	20.8	1.8
DFO trees parallelized	31	0.0	0.0
PX local messages recv'd	192,660	13.4	1.2
PX local messages sent	192,431	13.4	1.2
Parallel operations downgraded 1	0	0.0	0.0
Parallel operations downgraded 25	6	0.0	0.0
Parallel operations downgraded 50	0	0.0	0.0
Parallel operations downgraded to	2	0.0	0.0
Parallel operations not downgrade	25	0.0	0.0
SQL*Net roundtrips to/from client	22,221,687	1,543.6	133.0
SQL*Net roundtrips to/from dblink	19,350	1.3	0.1
background checkpoints completed	17	0.0	0.0
background checkpoints started	17	0.0	0.0
background timeouts	45,484	3.2	0.3
branch node splits	16	0.0	0.0
buffer is not pinned count	448,781,851	31,174.1	2,686.8
buffer is pinned count	7,069,302,643	491,060.2	42,322.3
bytes received via SQL*Net from c	7,237,120,160	502,717.4	43,327.0
bytes received via SQL*Net from d	1,112,038	77.3	6.7
bytes sent via SQL*Net to client	6,529,057,411	453,532.8	39,088.0
bytes sent via SQL*Net to dblink	243,692,274	16,927.8	1,458.9
calls to get snapshot scn: kcmgss	12,96[illegible]019	900.7	77.6
calls to kcmgas	162,786	11.3	1.0
calls to kcmgcs	237,905	16.5	1.4
change write time	34,321	2.4	0.2
cleanouts and rollbacks - consist	193,015	13.4	1.2
cleanouts only - consistent read	119,182	8.3	0.7
cluster key scan block gets	1,203,291	83.6	7.2
cluster key scans	458,032	31.8	2.7
commit cleanout failures: block l	103,076	7.2	0.6
commit cleanout failures: buffer	48,540	3.4	0.3
commit cleanout failures: callbac	1,958	0.1	0.0
commit cleanout failures: cannot	1,564	0.1	0.0
commit cleanouts	1,252,949	87.0	7.5
commit cleanouts successfully com	1,097,811	76.3	6.6

Instance Activity Stats for DB: dba01 Instance: dba01 Snaps: 2 -4

Statistic	Total	per Second	per Trans
consistent changes	704,266	48.9	4.2
consistent gets	612,237,120	42,528.3	3,665.3
current blocks converted for CR			
cursor authentications	9,490	0.7	0.1
data blocks consistent reads - un	685,485	47.6	4.1
db block changes	10,230,141	710.6	61.3
db block gets	17,972,677	1,248.5	107.6
deferred (CURRENT) block cleanout	251,171	17.5	1.5
dirty buffers inspected	99,512	6.9	0.6
enqueue conversions	20,795	1.4	0.1
enqueue releases	676,178	47.0	4.1
enqueue requests	690,640	48.0	4.1
enqueue timeouts	14,029	1.0	0.1
enqueue waits	14,033	1.0	0.1
exchange deadlocks	6	0.0	0.0
execute count	10,828,737	752.2	64.8
free buffer inspected	133,072	9.2	0.8
free buffer requested	30,749,604	2,136.0	184.1
hot buffers moved to head of LRU	3,656,215	254.0	21.9
immediate (CR) block cleanout app	312,198	21.7	1.9
immediate (CURRENT) block cleanou	77,797	5.4	0.5
index fast full scans (direct rea	0	0.0	0.0
index fast full scans (full)	1,310	0.1	0.0
index fast full scans (rowid rang	0	0.0	0.0
leaf node splits	5,576	0.4	0.0
logons cumulative	9,049	0.6	0.1
messages received	1,077,361	74.8	6.5
messages sent	1,077,363	74.8	6.5
no buffer to keep pinned count	106,061,351	7,367.4	635.0
no work - consistent read gets	279,634,470	19,424.5	1,674.1
opened cursors cumulative	226,594	15.7	1.4
opened cursors current			
parse count (hard)	26,557	1.8	0.2
parse count (total)	9,082,240	630.8	54.4
parse time cpu	110,501	7.7	0.7
parse time elapsed	153,495	10.7	0.9
physical reads	38,138,047	2,649.2	228.3
physical reads direct	8,131,985	564.9	48.7
physical writes	2,617,275	181.8	15.7
physical writes direct	130,381	9.1	0.8
physical writes non checkpoint	1,051,637	73.1	6.3
pinned buffers inspected	1,108	0.1	0.0
prefetched blocks	13,426,209	932.6	80.4
prefetched blocks aged out before	3,043	0.2	0.0
process last non-idle time	################	############	############
queries parallelized	31	0.0	0.0
recursive calls	2,909,676	202.1	17.4
recursive cpu usage	142,030	9.9	0.9

Instance Activity Stats for DB: dba01 Instance: dba01 Snaps: 2 -4

Statistic	Total	per Second	per Trans
redo blocks written	2,054,773	142.7	12.3
redo buffer allocation retries	71	0.0	0.0
redo entries	5,288,608	367.4	31.7
redo log space requests	76	0.0	0.0
redo log space wait time	578	0.0	0.0
redo ordering marks	0	0.0	0.0
redo size	1,911,814,688	132,801.8	11,445.6
redo synch time	62,483	4.3	0.4
redo synch writes	148,848	10.3	0.9
redo wastage	159,124,280	11,053.4	952.6
redo write time	88,292	6.1	0.5
redo writer latching time	327	0.0	0.0
redo writes	319,838	22.2	1.9
rollback changes - undo records a	19,210	1.3	0.1
rollbacks only - consistent read	24,812	1.7	0.2
rows fetched via callback	101,187,960	7,028.9	605.8
session connect time	################	############	############
session cursor cache count	2,727	0.2	0.0
session cursor cache hits	8,331,445	578.7	49.9
session logical reads	622,163,159	43,217.8	3,724.8
session pga memory	10,422,061,992	723,955.4	62,394.5
session pga memory max	9,945,785,568	690,871.5	59,543.1
session uga memory	66,411,464	4,613.2	397.6
session uga memory max	1,202,141,544	83,505.3	7,196.9
sorts (disk)	35	0.0	0.(
sorts (memory)	4,172,017	289.8	25.0
sorts (rows)	37,870,908	2,630.7	226.7
summed dirty queue length	18,049	1.3	0.1
switch current to new buffer			
table fetch by rowid	3,742,857,902	259,992.9	22,407.6
table fetch continued row	14,854	1.0	0.1
table scan blocks gotten	21,913,768	1,522.2	131.2
table scan rows gotten	972,327,781	67,541.5	5,821.1
table scans (direct read)	2,831	0.2	0.0
table scans (long tables)	3,986	0.3	0.0
table scans (rowid ranges)	################	############	############
table scans (short tables)	1,460,315	101.4	8.7
total file opens	78,945	5.5	0.5
transaction rollbacks	1,609	0.1	0.0
transaction tables consistent rea	60	0.0	0.0
transaction tables consistent rea	20,137	1.4	0.1
user calls	22,243,790	1,545.1	133.2
user commits	146,266	10.2	0.9
user rollbacks	20,769	1.4	0.1
write clones created in backgroun	2,222	0.2	0.0
write clones created in foregroun	206,401	14.3	1.2

Tablespace IO Stats for DB: dba01 Instance: dba01 Snaps: 2 -4
->ordered by IOs (Reads + Writes) desc

Tablespace

	Reads	Av Reads/s	Av Rd(ms)	Av Blks/Rd	Writes	Av Writes/s	Buffer Waits	Av Buf Wt(ms)
TBSL05	4,585,443	319	2.2	1.0	62,533	4	66,891	6.4
TBSL04	3,403,037	236	11.9	1.0	61,022	4	34,308	6.2
TBSM51	1,671,136	116	5.5	6.0	116,667	8	23,664	5.7
TBSL03	1,513,824	105	12.4	6.3	27,934	2	12,410	12.0
TBS0A	420,331	29	7.6	1.0	774,704	54	10,771	9.3
TBSL06	1,048,150	73	13.7	1.0	25,361	2	20	10.5
TBSM02	1,022,425	71	6.8	2.7	29,989	2	3,148	6.3
TBSM01	911,925	63	5.6	2.1	33,137	2	1,751	3.1
TBSM53	483,197	34	10.4	1.0	399,383	28	204	14.3
TBSS01	610,371	42	73.1	4.3	105,236	7	5,701	316.6
TBSS51	368,345	26	12.4	1.0	34,239	2	814	8.5
TBSL51	327,978	23	5.2	1.0	56,849	4	87	15.5
TBSL53	142,554	10	14.0	1.0	150,217	10	85	11.6
RBS1	27,336	2	7.2	1.0	209,775	15	224	1.4
TBSL52	135,013	9	23.6	1.0	99,892	7	1	0.0
RBS2	21,384	1	7.0	1.0	212,179	15	213	1.4
TBSL02	75,604	5	10.3	1.0	42,365	3	8	2.5
TBSM52	23,793	2	10.8	1.0	18,339	1	1	0.0
TBSL01	16,006	1	13.5	1.0	14,247	1	3	16.7
SYSTEM	17,413	1	8.6	2.7	6,757	0	30	1.0
TOOLS	11,865	1	2.4	1.3	1,499	0	1	0.0
TEMP1	3,770	0	0.0	22.3	5,267	0	0	0.0

```
Tablespace IO Stats for DB: dba01  Instance: dba01  Snaps: 2 -4
->ordered by IOs (Reads + Writes) desc

Tablespace
------------------------------
                 Av      Av     Av                    Av        Buffer Av Buf
         Reads Reads/s Rd(ms) Blks/Rd       Writes Writes/s      Waits Wt(ms)
-------------- ------- ------ ------- ------------ -------- ---------- ------
RBSLARGE2
           428       0    6.4     1.0        3,445        0          6    1.7
RBSLARGE1
           402       0    6.8     1.0        2,898        0          3    0.0
TBSSNP
           291       0   11.6     3.8           17        0          0    0.0
USERS
           104       0    8.2     1.0           38        0          0    0.0
TBS_BACK
            86       0   16.2     1.0           17        0          0    0.0
PRECISE_TBS
            17       0    0.0     1.0           17        0          0    0.0
          ------------------------------------------------------------
```

File IO Stats for DB: dba01 Instance: dba01 Snaps: 2 -4
->ordered by Tablespace, File

Tablespace	Filename	Reads	Av Reads/s	Av Rd(ms)	Av Blks/Rd	Writes	Av Writes/s	Buffer Waits	Av Buf Wt(ms)
PRECISE_TBS	/dba01/ORADATA/dbf/precise0.dbf	17	0	0.0	1.0	17	0	0	
RBS1	/dba01/ORADATA/dbf/rbs1dba01_1.dbf	27,336	2	7.2	1.0	209,775	15	224	1.4
RBS2	/dba01/ORADATA/dbf/rbs2dba01_1.dbf	21,384	1	7.0	1.0	212,179	15	213	1.4
RBSLARGE1	/dba01/ORADATA/dbf/rbslarge1dba01_1.dbf	402	0	6.8	1.0	2,898	0	3	0.0
RBSLARGE2	/dba01/ORADATA/dbf/rbslarge2dba01_1.dbf	428	0	6.4	1.0	3,445	0	6	1.7
SYSTEM	/dba01/ORADATA/dbf/systemdba01_1.dbf	12,030	1	8.3	2.6	5,906	0	30	1.0
	/dba01/ORADATA/dbf/systemdba01_2.dbf	5,413	0	9.2	3.0	851	0	0	
TBS0A	/dba01/ORADATA/dbf/tbs0adba01_1.dbf	27,610	2	8.3	1.0	62,721	4	253	4.2
	/dba01/ORADATA/dbf/tbs0adba01_2.dbf	320,242	22	7.3	1.0	563,844	39	10,301	9.6
	/dba01/ORADATA/dbf/tbs0adba01_3.dbf	46,608	3	8.6	1.0	97,013	7	62	2.1
	/dba01/ORADATA/dbf/tbs0adba01_4.dbf	25,871	2	8.7	1.0	51,126	4	155	5.4
TBSL01	/dba01/ORADATA/dbf/tbsl01dba01_2.dbf	3,959	0	13.7	1.0	17	0	0	
	/dba01/ORADATA/dbf/tbsl01dba01_3.dbf	3,209	0	13.9	1.0	792	0	0	
	/dba01/ORADATA/dbf/tbsl01dba01_4.dbf	4,981	0	11.4	1.0	13,421	1	3	16.7
	/dba01/ORADATA/dbf/tbsl01dba01_1.dbf	3,857	0	15.9	1.0	17	0	0	
TBSL02	/dba01/ORADATA/dbf/tbsl02dba01_1.dbf	41,395	3	10.2	1.0	30,308	2	8	2.5
	/dba01/ORADATA/dbf/tbsl02dba01_2.dbf	23,648	2	10.3	1.0	5,922	0	0	
	/dba01/ORADATA/dbf/tbsl02dba01_3.dbf	10,561	1	10.9	1.0	6,135	0	0	
TBSL03	/dba01/ORADATA/dbf/tbsl03dba01_1.dbf	577,583	40	15.2	6.5	933	0	2,479	13.6
	/dba01/ORADATA/dbf/tbsl03dba01_2.dbf	571,901	40	13.8	6.7	26	0	3,297	14.0
	/dba01/ORADATA/dbf/tbsl03dba01_3.dbf	364,340	25	5.8	5.4	26,975	2	6,634	10.4

File IO Stats for DB: dba01 Instance: dba01 Snaps: 2 -4
->ordered by Tablespace, File

Tablespace	Filename	Reads	Av Reads/s	Av Rd(ms)	Av Blks/Rd	Writes	Av Writes/s	Buffer Waits	Av Buf Wt(ms)
TBSL04	/dba01/ORADATA/dbf/tbsl04dba01_1.dbf	854,941	59	21.3	1.0	833	0	3,095	6.2
	/dba01/ORADATA/dbf/tbsl04dba01_3.dbf	1,125,051	78	4.7	1.0	59,924	4	7,362	3.7
	/dba01/ORADATA/dbf/tbsl04dba01_2.dbf	1,423,045	99	11.9	1.0	265	0	23,851	6.9
TBSL05	/dba01/ORADATA/dbf/tbsl05dba01_3.dbf	1,543,543	107	1.8	1.0	61,292	4	32	5.0
	/dba01/ORADATA/dbf/tbsl05dba01_1.dbf	1,924,576	134	2.1	1.0	345	0	24,043	6.0
	/dba01/ORADATA/dbf/tbsl05dba01_2.dbf	1,117,324	78	2.8	1.0	896	0	42,816	6.6
TBSL06	/dba01/ORADATA/dbf/tbsl06dba01_2.dbf	121,166	8	10.6	1.0	25,107	2	13	6.2
	/dba01/ORADATA/dbf/tbsl06dba01_1.dbf	926,984	64	14.1	1.0	254	0	7	18.6
TBSL51	/dba01/ORADATA/dbf/tbsl512dba01_2.dbf	91,297	6	8.0	1.0	42,160	3	47	15.1
	/dba01/ORADATA/dbf/tbsl512dba01_3.dbf	150,685	10	3.0	1.0	13,326	1	18	15.6
	/dba01/ORADATA/dbf/tbsl512dba01_1.dbf	85,996	6	5.8	1.0	1,363	0	2	25.0
TBSL52	/dba01/ORADATA/dbf/tbsl52dba01_2.dbf	47,152	3	36.7	1.0	28,202	2	9	
	/dba01/ORADATA/dbf/tbsl52dba01_1.dbf	87,861	6	16.5	1.0	70,690	5	1	0.0
TBSL53	/dba01/ORADATA/dbf/tbsl53dba01_1.dbf	46,492	3	11.5	1.0	31,670	2	12	10.8
	/dba01/ORADATA/dbf/tbsl53dba01_2.dbf	31,676	2	10.8	1.0	10,058	1	2	15.0
	/dba01/ORADATA/dbf/tbsl53dba01_3.dbf	64,386	4	17.4	1.0	99,489	7	71	11.7
TBSM01	/dba01/ORADATA/dbf/tbsm01dba01_2.dbf	43,083	3	7.9	2.4	12,864	1	19	71.6
	/dba01/ORADATA/dbf/tbsm01dba01_1.dbf	868,842	60	5.5	2.1	20,273	1	1,732	2.4
TBSM02	/dba01/ORADATA/dbf/tbsm02dba01_1.dbf	1,022,425	71	6.8	2.7	29,989	2	3,148	6.3
TBSM51	/dba01/ORADATA/dbf/tbsm51dba01_2.dbf	889,417	62	5.1	5.9	60,741	4	11,273	5.6
	/dba01/ORADATA/dbf/tbsm51dba01_1.dbf	731,709	54	5.9	6.1	55,926	4	12,391	5.7

```
File IO Stats for DB: dba01  Instance: dba01  Snaps: 2 -4
->ordered by Tablespace, File
Tablespace               Filename
------------------------ ----------------------------------------------------
                 Av      Av     Av                    Av        Buffer Av Buf
         Reads Reads/s Rd(ms) Blks/Rd       Writes Writes/s      Waits Wt(ms)
-------------- ------- ------ ------- ------------ -------- ---------- ------
TBSM52                   /dba01/ORADATA/dbf/tbsm52dba01_1.dbf
        16,649       1   10.1     1.0       11,222        1          1    0.0
TBSM52                   /dba01/ORADATA/dbf/tbsm52dba01_2.dbf
         7,144       0   12.4     1.0        7,117        0          0
TBSM53                   /dba01/ORADATA/dbf/tbsm53dba01_3.dbf
       240,303      17    8.8     1.0      338,846       24         67   14.6
                         /dba01/ORADATA/dbf/tbsm53dba01_1.dbf
       128,858       9   12.0     1.0       28,166        2        111   12.9
                         /dba01/ORADATA/dbf/tbsm53dba01_2.dbf
       114,036       8   11.9     1.0       32,371        2         26   19.6
TBSS01                   /dba01/ORADATA/dbf/tbss01dba01_1.dbf
       610,371      42   73.1     4.3      105,236        7      5,701  316.6
TBSS51                   /dba01/ORADATA/dbf/tbss51dba01_1.dbf
       368,345      26   12.4     1.0       34,239        2        814    8.5
TBSSNP                   /dba01/ORADATA/dbf/tbssnp_1.dbf
           291       0   11.6     3.8           17        0          0
TBS_BACK                 /dba01/ORADATA/dbf/tbs_backdba01_1.dbf
            86       0   16.2     1.0           17        0          0
TEMP1                    /dba01/ORADATA/dbf/temp1dba01_1.dbf
            34       0    0.0     1.0           17        0          0
                         /dba01/ORADATA/dbf/temp1dba01_2.dbf
         3,736       0    0.0    22.5        5,250        0          0
TOOLS                    /dba01/ORADATA/dbf/toolsdba01_1.dbf
        11,207       1    2.2     1.3        1,137        0          1    0.0
                         /dba01/ORADATA/dbf/toolsdba01_2.dbf
           658       0    4.4     1.0          362        0          0

USERS                    /dba01/ORADATA/dbf/usersdba01_1.dbf
            87       0    9.8     1.0           21        0          0
                         /dba01/ORADATA/dbf/usersdba01_2.dbf
            17       0    0.0     1.0           17        0          0
          -------------------------------------------------------------
```

Buffer Pool Statistics for DB: dba01 Instance: dba01 Snaps: 2 -4
-> Pools   D: default pool, K: keep pool, R: recycle pool

P	Buffer Gets	Consistent Gets	Physical Reads	Physical Writes	Free Buffer Waits	Write Complete Waits	Buffer Busy Waits
D	30,754,181	0	30,010,736	2,486,800	0	0	160,335

Buffer wait Statistics for DB: dba01 Instance: dba01 Snaps: 2 -4
-> ordered by wait time desc, waits desc

Class	Waits	Tot Wait Time (cs)	Avg Time (cs)
data block	159,656	286,451	2
segment header	226	100	0
undo header	326	40	0
undo block	120	22	0

Enqueue activity for DB: dba01 Instance: dba01 Snaps: 2 -4
-> ordered by waits desc, gets desc

Enqueue	Gets	Waits
TX	177,029	14,031

Rollback Segment Stats for DB: dba01 Instance: dba01 Snaps: 2 -4
->A high value for "Pct Waits" suggests more rollback segments may be required

RBS No	Trans Table Gets	Pct Waits	Undo Bytes Written	Wraps	Shrinks	Extends
0	100.0	0.00	0	0	0	0
76	4,009.0	0.00	3,011,846	0	0	0
77	4,274.0	0.00	4,036,866	0	0	0
79	5,010.0	0.00	4,279,146	3	0	0
80	4,730.0	0.00	4,180,974	2	0	0
81	3,418.0	0.00	3,087,764	2	0	0
82	7,077.0	0.00	6,523,904	4	0	0
83	4,798.0	0.00	4,159,058	2	0	0
84	4,757.0	0.02	3,809,838	2	0	0
85	5,005.0	0.00	4,020,058	2	0	0
86	4,460.0	0.02	3,950,066	2	0	0
87	5,175.0	0.04	5,554,790	3	0	0
88	5,032.0	0.02	5,355,284	3	0	0
89	5,244.0	0.00	4,220,486	2	0	0
90	4,350.0	0.02	3,973,104	2	0	0
91	9,388.0	0.00	31,638,132	15	0	0
92	5,800.0	0.00	5,369,852	2	0	0
93	5,743.0	0.00	5,332,320	2	0	0
94	5,333.0	0.00	15,988,656	8	0	0
95	5,063.0	0.00	3,859,788	2	0	0
96	3,559.0	0.00	3,028,792	2	0	0
97	5,482.0	0.02	5,431,612	3	0	0
98	4,020.0	0.00	2,925,590	2	0	0
99	4,072.0	0.00	2,928,528	1	0	0
100	2,941.0	0.00	1,801,368	1	0	0
101	7,341.0	0.03	7,861,032	4	0	0
102	4,651.0	0.02	4,961,230	2	0	0
103	5,536.0	0.00	4,666,606	2	0	0
104	4,048.0	0.02	3,630,112	1	0	0
105	4,602.0	0.00	5,101,026	2	0	0
106	12,514.0	0.01	69,910,272	34	0	15
107	3,982.0	0.00	2,917,216	2	0	0
108	4,548.0	0.02	3,689,526	2	0	0
109	4,450.0	0.00	3,408,134	1	0	0
110	3,694.0	0.00	3,163,236	1	0	0
111	10,235.0	0.02	12,542,386	16	0	0
112	4,149.0	0.00	3,026,554	1	0	0
113	4,771.0	0.02	3,423,674	2	0	0
114	4,917.0	0.00	3,505,344	2	0	0
115	4,003.0	0.00	4,003,340	2	0	0
116	5,211.0	0.00	4,240,924	2	0	0
117	5,580.0	0.02	4,637,814	3	0	0
118	4,654.0	0.00	3,164,486	1	0	0
119	4,769.0	0.00	3,884,436	2	0	0

Rollback Segment Stats for DB: dba01 Instance: dba01 Snaps: 2 -4
->A high value for "Pct Waits" suggests more rollback segments may be required

RBS No	Trans Table Gets	Pct Waits	Undo Bytes Written	Wraps	Shrinks	Extends
120	4,021.0	0.00	3,034,092	1	0	0
121	4,039.0	0.00	3,191,548	2	0	0
122	3,870.0	0.00	3,778,458	2	0	0
123	3,644.0	0.00	3,018,332	2	0	0
124	3,191.0	0.00	2,697,214	2	0	0
125	2,967.0	0.00	2,088,206	1	0	0
126	4,149.0	0.00	3,454,876	2	0	0
127	3,975.0	0.00	3,879,908	2	0	0
128	5,836.0	0.00	4,917,940	2	0	0
129	4,245.0	0.00	3,963,266	2	0	0
130	6,379.0	0.00	6,483,112	3	0	0
132	12,307.0	0.02	69,432,430	34	0	15
133	14,715.0	0.02	76,919,156	38	0	17
134	3,828.0	0.00	3,213,020	2	0	0
135	5,614.0	0.04	4,922,670	2	0	0
136	5,114.0	0.04	3,676,814	2	0	0
137	3,419.0	0.03	2,525,042	2	0	0
138	6,949.0	0.01	6,341,890	4	0	0
139	4,328.0	0.02	4,086,654	2	0	0
140	4,767.0	0.00	4,266,816	2	0	0
141	3,859.0	0.03	3,044,358	1	0	0
142	3,712.0	0.00	3,077,798	1	0	0
143	5,535.0	0.02	16,545,146	8	0	0
144	4,025.0	0.02	3,053,298	1	0	0
145	5,772.0	0.00	5,768,094	3	0	0
146	4,597.0	0.00	5,101,434	3	0	0
147	5,234.0	0.02	4,554,324	2	0	0
148	5,064.0	0.00	4,216,938	3	0	0
149	5,836.0	0.00	6,692,288	4	0	0
150	4,388.0	0.00	4,262,966	3	0	0
151	3,653.0	0.00	2,727,238	2	0	0
152	4,589.0	0.00	5,336,590	2	0	0
153	5,316.0	0.00	4,030,252	2	0	0
154	4,843.0	0.00	4,235,534	2	0	0
155	5,598.0	0.00	16,849,224	9	0	0
156	5,245.0	0.02	15,931,224	7	0	2
157	3,502.0	0.03	2,713,450	1	0	0
158	3,909.0	0.00	2,782,836	2	0	0
159	4,628.0	0.00	3,758,808	2	0	0
160	7,436.0	0.00	27,300,394	13	0	0
161	4,897.0	0.00	11,305,832	5	0	0
162	4,899.0	0.00	3,819,822	2	0	0
163	4,417.0	0.00	3,056,898	2	0	0
164	4,052.0	0.02	3,256,102	2	0	0
165	3,972.0	0.00	3,645,832	2	0	0

Rollback Segment Stats for DB: dba01 Instance: dba01 Snaps: 2 -4
->A high value for "Pct Waits" suggests more rollback segments may be required

RBS No	Trans Table Gets	Pct Waits	Undo Bytes Written	Wraps	Shrinks	Extends
166	5,505.0	0.00	5,354,352	3	0	0
167	5,949.0	0.00	16,781,928	8	0	0
168	3,713.0	0.00	3,255,552	1	0	0
169	5,009.0	0.02	4,150,532	2	0	0
170	5,466.0	0.02	4,449,518	2	0	0
171	3,958.0	0.00	3,185,946	1	0	0
172	3,355.0	0.00	2,636,056	2	0	0
173	4,568.0	0.00	4,095,850	2	0	0
174	3,830.0	0.00	4,194,192	2	0	0
175	3,895.0	0.00	3,489,200	2	0	0
176	5,780.0	0.00	16,842,136	8	0	0
177	3,969.0	0.00	3,334,172	2	0	0
178	4,636.0	0.00	3,859,346	2	0	0
179	4,290.0	0.00	3,641,736	2	0	0

Rollback Segment Storage for DB: dba01 Instance: dba01 Snaps: 2 -4
->Optimal Size should be larger than Avg Active

RBS No	Segment Size	Avg Active	Optimal Size	Maximum Size
0	1,138,688	7,372		1,138,688
76	419,422,208	49,131,373	419,430,400	419,422,208
77	419,422,208	0	419,430,400	419,422,208
79	42,590,208	4,297,880	41,943,040	42,590,208
80	42,590,208	5,130,462	41,943,040	42,590,208
81	42,590,208	5,508,272	41,943,040	42,590,208
82	42,590,208	5,646,238	41,943,040	42,590,208
83	42,590,208	3,806,628	41,943,040	42,590,208
84	42,590,208	3,065,805	41,943,040	42,590,208
85	42,590,208	5,637,723	41,943,040	42,590,208
86	42,590,208	3,437,246	41,943,040	42,590,208
87	42,590,208	2,635,135	41,943,040	42,590,208
88	42,590,208	2,241,796	41,943,040	42,590,208
89	42,590,208	1,993,453	41,943,040	42,590,208
90	42,590,208	1,884,079	41,943,040	42,590,208
91	42,590,208	13,249,096	41,943,040	42,590,208
92	42,590,208	2,496,391	41,943,040	42,590,208
93	42,590,208	4,960,473	41,943,040	42,590,208
94	42,590,208	7,398,537	41,943,040	42,590,208
95	42,590,208	2,788,505	41,943,040	42,590,208
96	42,590,208	3,681,341	41,943,040	42,590,208
97	42,590,208	3,620,442	41,943,040	42,590,208
98	42,590,208	2,747,634	41,943,040	42,590,208
99	42,590,208	2,922,087	41,943,040	42,590,208
100	42,590,208	4,692,974	41,943,040	42,590,208
101	42,590,208	4,897,030	41,943,040	42,590,208
102	42,590,208	2,708,323	41,943,040	42,590,208
103	42,590,208	2,083,221	41,943,040	42,590,208
104	42,590,208	8,305,261	41,943,040	42,590,208
105	42,590,208	5,268,467	41,943,040	42,590,208
106	74,539,008	53,892,383	41,943,040	74,539,008
107	42,590,208	7,586,007	41,943,040	42,590,208
108	42,590,208	6,615,188	41,943,040	42,590,208
109	42,590,208	6,571,361	41,943,040	42,590,208
110	42,590,208	6,016,008	41,943,040	42,590,208
111	42,590,208	17,454,199	41,943,040	42,590,208
112	42,590,208	2,285,771	41,943,040	42,590,208
113	42,590,208	3,952,768	41,943,040	42,590,208
114	42,590,208	1,858,952	41,943,040	42,590,208
115	42,590,208	9,023,487	41,943,040	42,590,208
116	42,590,208	4,732,644	41,943,040	42,590,208
117	42,590,208	6,813,383	41,943,040	42,590,208
118	42,590,208	8,513,858	41,943,040	42,590,208
119	42,590,208	4,735,765	41,943,040	42,590,208
120	42,590,208	21,716,974	41,943,040	51,109,888
121	42,590,208	1,964,246	41,943,040	42,590,208
122	42,590,208	217,693,625	41,943,040	511,172,608

Rollback Segment Storage for DB: dba01 Instance: dba01 Snaps: 2 -4
->Optimal Size should be larger than Avg Active

RBS No	Segment Size	Avg Active	Optimal Size	Maximum Size
123	42,590,208	5,718,693	41,943,040	42,590,208
124	42,590,208	4,443,753	41,943,040	42,590,208
125	42,590,208	2,435,736	41,943,040	42,590,208
126	42,590,208	2,643,884	41,943,040	42,590,208
127	42,590,208	7,561,782	41,943,040	46,850,048
128	42,590,208	2,465,094	41,943,040	42,590,208
129	42,590,208	2,566,604	41,943,040	42,590,208
130	42,590,208	3,693,499	41,943,040	42,590,208
132	74,539,008	53,827,217	41,943,040	74,539,008
133	78,798,848	59,264,629	41,943,040	78,798,848
134	42,590,208	5,467,496	41,943,040	51,109,888
135	42,590,208	13,483,036	41,943,040	46,850,048
136	42,590,208	12,899,828	41,943,040	42,590,208
137	42,590,208	4,609,633	41,943,040	42,590,208
138	42,590,208	3,637,713	41,943,040	42,590,208
139	42,590,208	2,491,798	41,943,040	42,590,208
140	42,590,208	7,321,185	41,943,040	42,590,208
141	42,590,208	4,909,855	41,943,040	42,590,208
142	42,590,208	13,213,714	41,943,040	48,979,968
143	42,590,208	9,388,604	41,943,040	42,590,208
144	42,590,208	9,073,568	41,943,040	42,590,208
145	42,590,208	6,221,551	41,943,040	42,590,208
146	42,590,208	4,588,554	41,943,040	42,590,208
147	42,590,208	2,342,772	41,943,040	42,590,208
148	42,590,208	4,365,806	41,943,040	42,590,208
149	42,590,208	5,018,033	41,943,040	42,590,208
150	42,590,208	4,579,270	41,943,040	42,590,208
151	42,590,208	7,870,376	41,943,040	46,850,048
152	42,590,208	6,982,647	41,943,040	42,590,208
153	42,590,208	19,944,820	41,943,040	68,149,248
154	42,590,208	9,829,923	41,943,040	42,590,208
155	42,590,208	5,274,796	41,943,040	42,590,208
156	46,850,048	27,978,008	41,943,040	46,850,048
157	42,590,208	11,577,743	41,943,040	42,590,208
158	42,590,208	5,274,989	41,943,040	42,590,208
159	42,590,208	4,877,205	41,943,040	42,590,208
160	42,590,208	10,[illegible]935	41,943,040	42,590,208
161	42,590,208	3,769,022	41,943,040	42,590,208
162	42,590,208	8,625,142	41,943,040	51,109,888
163	42,590,208	4,548,788	41,943,040	42,590,208
164	42,590,208	6,881,776	41,943,040	42,590,208
165	42,590,208	4,090,556	41,943,040	42,590,208
166	42,590,208	8,519,558	41,943,040	42,590,208
167	42,590,208	9,969,199	41,943,040	42,590,208
168	42,590,208	2,372,896	41,943,040	42,590,208
169	42,590,208	2,259,582	41,943,040	42,590,208
170	42,590,208	8,825,690	41,943,040	48,979,968

```
Rollback Segment Storage for DB: dba01  Instance: dba01  Snaps: 2 -4
->Optimal Size should be larger than Avg Active

RBS No    Segment Size      Avg Active    Optimal Size    Maximum Size
------ --------------- --------------- --------------- ---------------
   171      42,590,208       6,191,750      41,943,040      42,590,208
   172      42,590,208      11,510,328      41,943,040      42,590,208
   173      42,590,208       7,715,760      41,943,040      42,590,208
   174      42,590,208       2,461,638      41,943,040      42,590,208
   175      42,590,208       6,116,715      41,943,040      42,590,208
   176      42,590,208       6,379,282      41,943,040      42,590,208
   177      42,590,208      15,872,040      41,943,040      63,889,408
   178      42,590,208       5,381,336      41,943,040      42,590,208
   179      42,590,208       7,941,825      41,943,040      42,590,208
          -------------------------------------------------------------
```

Latch Activity for DB: dba01 Instance: dba01 Snaps: 2 -4
->"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

Latch Name	Get Requests	Pct Get Miss	Avg Slps /Miss	NoWait Requests	Pct NoWait Miss
Token Manager	78,427	0.0	0.3	2,596	0.0
X$KSFQP	8	0.0		0	
active checkpoint queue latch	791,889	0.1	0.3	0	
archive control	274	0.0		0	
archive process latch	34	0.0		0	
begin backup scn array	10	0.0		0	
cache buffer handles	592,279,666	7.1	0.0	0	
cache buffers chains	984,933,131	0.1	0.3	44,678,716	0.1
cache buffers lru chain	9,553,732	0.2	0.5	30,330,326	0.2
channel handle pool latch	11,379	0.0		11,562	0.0
channel operations parent lat	19,532	0.0		11,562	0.0
checkpoint queue latch	29,410,803	0.1	0.1	0	
dictionary lookup	63	0.0		0	
dml lock allocation	753,125	0.0	0.1	0	
enqueue hash chains	1,875,817	0.1	0.7	0	
enqueues	1,425,986	0.0	0.1	0	
error message lists	1,031	3.9	0.1	0	
event group latch	8,153	0.0		0	
file number translation table	33	0.0		0	
global transaction	53,851	0.0		0	
global tx free list	2,532	0.0		0	
global tx hash mapping	19,572	0.0		0	
job_queue_processes parameter	250	0.0		0	
ktm global data	48	0.0		0	
latch wait list	141,415	0.6	0.3	141,300	0.2
library cache	68,488,577	0.6	0.6	80,409	1.3
library cache load lock	4,216	0.0		0	
list of block allocation	386,860	0.0	0.4	0	
loader state object freelist	6,126	0.1	0.0	0	
longop free list	1,325,432	0.6	0.0	0	
messages	3,778,456	0.3	0.1	0	
mostly latch-free SCN	490,890	0.0	0.0	0	
multiblock read objects	1,708,086	0.0	0.1	0	
ncodef allocation latch	250	0.0		0	
parallel query alloc buffer	14,696	20.2	0.3	0	
parallel query stats	632	29.3	1.8	0	
process allocation	8,153	0.1	1.0	8,153	0.0
process group creation	16,403	0.0		0	
process queue	5,099	1.1	0.3	0	
process queue reference	4,020,210	0.0	0.1	196,962	7.9
query server freelists	9,226	5.1	0.1	0	
query server process	86	0.0		86	0.0

```
Latch Activity for DB: dba01  Instance: dba01  Snaps: 2 -4
->"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
```

Latch Name	Get Requests	Pct Get Miss	Avg Slps /Miss	NoWait Requests	Pct NoWait Miss
redo allocation	5,928,960	0.1	0.1	0	
redo writing	2,092,841	0.4	0.4	0	
row cache objects	8,936,427	0.0	0.3	2,590	0.2
sequence cache	68,942	0.0	1.0	0	
session allocation	480,695	0.1	0.8	0	
session idle bit	44,701,432	0.0	0.1	0	
session switching	265	0.0		0	
shared pool	5,458,933	0.3	0.6	0	
sort extent pool	257	0.0		0	
transaction allocation	1,191,981	0.0	0.1	0	
transaction branch allocation	19,941	0.0		0	
undo global data	2,376,711	0.1	0.6	0	
user lock	33,572	0.0	0.5	0	

Latch Sleep breakdown for DB: dba01 Instance: dba01 Snaps: 2 -4
-> ordered by misses desc

Latch Name	Get Requests	Misses	Sleeps	Spin & Sleeps 1->4
cache buffer handles	592,279,666	42,157,280	1,013,749	41179902/942959/32745/1674/0
cache buffers chains	984,933,131	1,331,118	384,653	1013733/258073/52615/6697/0
library cache	68,488,577	341,875	192,024	215804/69669/48519/7883/0
checkpoint queue latch	29,410,803	16,889	1,747	15149/1733/7/0/0
cache buffers lru chain	9,553,732	14,343	7,191	7195/7107/40/1/0
shared pool	5,458,933	14,057	8,674	8554/2865/2429/209/0
messages	3,778,456	9,750	1,134	8646/1074/30/0/0
longop free list	1,328,432	8,521	250	8273/246/2/0/0
redo writing	2,092,841	7,903	3,199	4770/3075/54/4/0
redo allocation	5,928,960	6,633	817	5852/745/36/0/0
parallel query alloc buffe	14,696	2,969	960	2198/606/142/23/0
undo global data	2,376,711	2,827	1,643	1234/1543/50/0/0
session idle bit	44,701,432	2,166	238	1937/220/9/0/0
enqueue hash chains	1,875,817	1,021	694	336/678/6/1/0
row cache objects	8,936,427	849	283	570/275/4/0/0
active checkpoint queue la	791,859	836	267	579/247/10/0/0
latch wait list	141,415	802	220	658/70/72/2/0
enqueues	1,425,986	632	87	551/75/6/0/0
multiblock read objects	1,708,086	540	59	481/59/0/0/0
query server freelists	9,226	475	57	423/48/3/1/0
session allocation	480,695	442	345	198/180/38/26/0
transaction allocation	1,191,981	400	34	372/23/4/1/0
process queue reference	4,020,210	359	41	320/37/2/0/0

Latch Sleep breakdown for DB: dba01 Instance: dba01 Snaps: 2 -4
-> ordered by misses desc

Latch Name	Get Requests	Misses	Sleeps	Spin & Sleeps 1->4
parallel query stats	632	185	328	18/78/43/46/0
dml lock allocation	753,125	124	16	109/14/1/0/0
process queue	5,099	57	17	45/7/5/0/0
error message lists	1,031	40	3	37/3/0/0/0
mostly latch-free SCN	490,890	34	1	33/1/0/0/0
list of block allocation	386,860	31	11	21/9/1/0/0
process allocation	8,153	12	12	0/12/0/0/0
Token Manager	78,427	4	1	3/1/0/0/0
sequence cache	68,942	2	2	0/2/0/0/0
user lock	33,572	2	1	1/1/0/0/0

Latch Miss Sources for DB: dba01 Instance: dba01 Snaps: 2 -4
-> only latches with sleeps are shown
-> ordered by name, sleeps desc

Latch Name	Where	NoWait Misses	Sleeps	Waiter Sleeps
Token Manager	kgklookup	0	1	1
active checkpoint queue	kcbbacq: scan active check	0	266	267
active checkpoint queue	kcbstc1: Start a new check	0	1	0
cache buffer handles	kcbzgs	0	691,673	#######
cache buffer handles	kcbzfs	0	321,247	#######
cache buffers chains	kcbgtcr: kslbegin	0	356,732	#######
cache buffers chains	kcbzib: multi-block read:	0	7,768	0
cache buffers chains	kcbgcur: kslbegin	0	4,399	4,049
cache buffers chains	kcbrls: kslbegin	0	3,449	36,008
cache buffers chains	kcbget: pin buffer	0	3,040	2,723
cache buffers chains	kcbchg: kslbegin: bufs not	0	1,836	2,403
cache buffers chains	kcbgtcr	0	1,768	49
cache buffers chains	kcbbxsv	0	1,435	1,940
cache buffers chains	kcbbwb1	0	1,270	2,634
cache buffers chains	kcbzwb	0	1,133	628
cache buffers chains	kcbzgb: scan from tail. no	0	785	0
cache buffers chains	kcbnlc	0	515	661
cache buffers chains	kcbzib: finish free bufs	0	239	9,539
cache buffers chains	kcbchg: kslbegin: call CR	0	147	943
cache buffers chains	kcbzsc	0	61	8
cache buffers chains	kcbget: exchange rls	0	23	123
cache buffers chains	kcbget: exchange	0	21	295
cache buffers chains	kcbzib: exchange rls	0	8	41
cache buffers chains	kcbbwdb	0	7	1,2[illegible]
cache buffers chains	kcbchg: no fast path	0	4	17
cache buffers chains	kcbesc: escalate	0	3	0
cache buffers chains	kcbcge	0	1	27
cache buffers chains	kcbso1: set no access	0	1	15
cache buffers lru chain	kcbzgb: multiple sets nowa	0	6,359	0
cache buffers lru chain	kcbbiop: lru scan	0	545	84
cache buffers lru chain	kcbzgb: posted for free bu	0	212	433
cache buffers lru chain	kcbzar: KSLNBEGIN	0	56	4,892
cache buffers lru chain	kcbzgm	0	9	365
cache buffers lru chain	kcbbioc: age write clones	0	5	167
cache buffers lru chain	kcbbioc	0	3	1,025
cache buffers lru chain	kcbbxsv: move to being wri	0	1	109
cache buffers lru chain	kcbgtcr:CR Scan:KCBRSKIP	0	1	51
checkpoint queue latch	kcbklrba: compute lowest	0	728	1,430
checkpoint queue latch	kcbk0rrd: update recovery	0	348	3
checkpoint queue latch	kcbklbc: Link buffer into	0	310	144
checkpoint queue latch	kcbbxsv: move to being wri	0	142	29
checkpoint queue latch	kcbbwthc: thread checkpoin	0	111	39
checkpoint queue latch	kcbbcrcv: check recovery q	0	74	71
checkpoint queue latch	kcbnlc: Link buffers into	0	22	8
checkpoint queue latch	kcbswcu: Switch buffers	0	11	23
checkpoint queue latch	kcbbwtsc: file queues	0	1	0

Latch Miss Sources for DB: dba01 Instance: dba01 Snaps: 2 -4
-> only latches with sleeps are shown
-> ordered by name, sleeps desc

Latch Name	Where	NoWait Misses	Sleeps	Waiter Sleeps
cost function	kzulrl	0	1	1
dml lock allocation	ktaiam	0	10	10
dml lock allocation	ktaidm	0	6	6
enqueue hash chains	ksqcmi: get hash chain lat	0	486	288
enqueue hash chains	ksqgtl3	0	184	245
enqueue hash chains	ksqrcl	0	23	157
enqueue hash chains	ksqcnl	0	1	2
enqueues	ksqgtl2	0	38	21
enqueues	ksqgel: create enqueue	0	20	21
enqueues	ksqdel	0	12	3
enqueues	ksqrcl	0	10	21
enqueues	ksqies	0	4	18
enqueues	ksqgel: failed to get enqu	0	3	3
error message lists	kxfpqidqr2: KSLBEGIN	0	2	1
error message lists	kxfpqsnd	0	1	2
latch wait list	kslfre	90	159	220
latch wait list	kslges	134	61	0
library cache	kglpnal: child: alloc spac	0	68,765	50,847
library cache	kglhdgn: child:	0	64,595	11,806
library cache	kglupc: child	0	15,536	57,068
library cache	kglpnal: child: before pro	0	13,866	16,187
library cache	kglhdgc: child:	0	8,352	5,193
library cache	kglpnc: child	0	6,325	14,681
library cache	kgllkdl: child: cleanup	0	4,892	5,716
library cache	kglidp: parent	0	1,791	3
library cache	kglpnp: child	0	1,450	9,315
library cache	kgllkdl: child: free pin	0	1,141	11,290
library cache	kglic	0	869	72
library cache	kglget: child: KGLDSBYD	0	532	5,168
library cache	kglpin	0	391	1,738
library cache	kgldti: 2child	0	356	295
library cache	kglget: child: KGLDSBPD	0	241	34
library cache	kglobpn: child:	0	135	473
library cache	kglpnal: child: check gran	0	111	100
library cache	kglati	0	91	13
library cache	kglpnd: parent: purge	0	23	4
library cache	kglobld: child:	0	18	193
library cache	kgldrp: parent	0	12	1
library cache	kgldte: child 0	0	8	683
library cache	kgldnp: child	0	7	64
library cache	kglpnal: parent held, no p	0	7	0
library cache	kgldte: child	0	2	1
library cache	kgldtld: 2child	0	1	0
checkpoint queue latch	kcbswcu: Switch buffers	0	11	23
checkpoint queue latch	kcbbwtsc: file queues	0	1	0
cost function	kzulrl	0	1	1

Latch Miss Sources for DB: dba01 Instance: dba01 Snaps: 2 -4
-> only latches with sleeps are shown
-> ordered by name, sleeps desc

Latch Name	Where	NoWait Misses	Sleeps	Waiter Sleeps
dml lock allocation	ktaiam	0	10	10
dml lock allocation	ktaidm	0	6	6
enqueue hash chains	ksqcmi: get hash chain lat	0	486	288
enqueue hash chains	ksqgtl3	0	184	245
enqueue hash chains	ksqrcl	0	23	157
enqueue hash chains	ksqcnl	0	1	2
enqueues	ksqgtl2	0	38	21
enqueues	ksqgel: create enqueue	0	20	21
enqueues	ksqdel	0	12	3
enqueues	ksqrcl	0	10	21
enqueues	ksqies	0	4	18
enqueues	ksqgel: failed to get enqu	0	3	3
error message lists	kxfpqidqr2: KSLBEGIN	0	2	1
error message lists	kxfpqsnd	0	1	2
latch wait list	kslfre	90	159	220
latch wait list	kslges	134	61	0
library cache	kglpnal: child: alloc spac	0	68,765	50,847
library cache	kglhdgn: child:	0	64,595	11,806
library cache	kglupc: child	0	15,536	57,068
library cache	kglpnal: child: before pro	0	13,866	16,187
library cache	kglhdgc: child:	0	8,352	5,193
library cache	kglpnc: child	0	6,325	14,381
library cache	kgllkdl: child: cleanup	0	4,892	5,716
library cache	kglidp: parent	0	1,791	2
library cache	kglpnp: child	0	1,450	9,315
library cache	kgllkdl: child: free pin	0	1,141	11,290
library cache	kglic	0	369	72
library cache	kglget: child: KGLDSBYD	0	532	5,168
library cache	kglpin	0	391	1,738
library cache	kgldti: 2child	0	356	295
library cache	kglget: child: KGLDSBRD	0	241	34
library cache	kglobpn: child:	0	135	473
library cache	kglpnal: child: check gran	0	111	100
library cache	kglati	0	91	13
library cache	kglpndl: parent: purge	0	23	4
library cache	kglobu: child:	0	18	193
library cache	kgldrp: parent	0	12	1
library cache	kgldte: child 0	0	8	683
library cache	kgldnp: child	0	7	64
library cache	kglpnal: parent held, no p	0	7	0
library cache	kgldte: child	0	2	1
library cache	kgldtld: 2child	0	1	0
library cache	kglpin: child: KGLMX	0	1	0
list of block allocation	ktlabl	0	7	5
list of block allocation	ktlbbl	0	4	6
longop free list	ksuloget	0	250	250

```
Latch Miss Sources for DB: dba01  Instance: dba01  Snaps: 2 -4
-> only latches with sleeps are shown
-> ordered by name, sleeps desc
                                                    NoWait              Waiter
Latch Name               Where                      Misses     Sleeps   Sleeps
------------------------ -------------------------- ------- ---------- -------
messages                 ksaamb: after wakeup            0        569     811
messages                 ksarcv: after wait              0        389     193
messages                 ksarcv                          0        175     125
messages                 ksaclr                          0          1       3
mostly latch-free SCN    kcs02                           0          1       1
multiblock read objects  kcbzib: MBRGET                  0         33      35
multiblock read objects  kcbzib: MBRFRE                  0         26      24
parallel query alloc buf kxfpbalo                        0        565     559
parallel query alloc buf kxfpbfre                        0        394     401
parallel query stats     kxfprst: KSLBEGIN               0        328     328
process allocation       ksuapc                          0         12      12
query server freelists   kxfpqrsnd                       0         41       0
query server freelists   kxfpobadf                       0         37      37
query server freelists   kxfpobrmf                       0         20      20
query server freelists   kxfpqsnd: KSLBEGIN              0         11      13
query server freelists   kxfpqidqr1: KSLBEGIN            0          6       4
redo allocation          kcrfwr: redo allocation         0        541     712
redo allocation          kcrfwi: before write            0        233      50
redo allocation          kcrfwi: more space              0         43      55
redo writing             kcrfsr                          0      2,872      11
redo writing             kcrfwi: after write             0        163      44
redo writing             kcrfwcr                         0         88     813
redo writing             kcrfss                          0         76   2,830
row cache objects        kqrpre: find obj                0        252     101
row cache objects        kqreqd: rel enqueue             0         13      30
row cache objects        kqrso                           0          0       1
row cache objects        kqreqd                          0          4      86
row cache objects        kqrpsc: incr stat               0          2       0
sequence cache           kdnnxt: cached seq              0          2       0
session allocation       ksuxds: KSUSFCLC not set        0        260     173
session allocation       kxfpqidqr                       0         50      38
session allocation       ksucri                          0         23      47
session allocation       kspsi                           0          4      45
session allocation       ksusin                          0          4       2
session allocation       ksuxds: not user session        0          4      25
session idle bit         ksupuc: clear busy              0        118      78
session idle bit         ksupuc: set busy                0        114     159
session idle bit         ksuxds                          0          6       1
shared pool              kghfrunp: alloc: clatch no      0      3,149       0
shared pool              kghalo                          0      2,626   1,620
shared pool              kghfrunp: clatch: nowait        0      2,282       0
shared pool              kghupr1                         0      1,214   6,125
shared pool              kghfre                          0        975     644
shared pool              kghfrunp: alloc: wait           0        339      38
shared pool              kghfrunp: clatch: wait          0        220   1,372
shared pool              kghfnd: min scan                0        176       0
```

```
Latch Miss Sources for DB: dba01  Instance: dba01  Snaps: 2 -4
-> only latches with sleeps are shown
-> ordered by name, sleeps desc
                                                    NoWait              Waiter
Latch Name               Where                      Misses     Sleeps   Sleeps
------------------------ -------------------------- ------- ---------- -------
shared pool              kghfen: not perm alloc cla       0         52      94
shared pool              kghfnd: req scan                 0         49       0
shared pool              kghalp                           0         41     145
shared pool              kghfnd: get next extent          0         36       0
shared pool              kghfrunp: no latch               0         10       0
shared pool              kghfru                           0          7       8
transaction allocation   ktcxba                           0         30      20
transaction allocation   ktcdso                           0          4      14
undo global data         ktubnd                           0      1,247     152
undo global data         ktudba: KSLBEGIN                 0        323   1,302
undo global data         ktudnx: KSLBEGIN                 0         70     186
undo global data         ktucof: at start                 0          3       0
          -------------------------------------------------------------

Dictionary Cache Stats for DB: dba01  Instance: dba01  Snaps: 2 -4
->"Pct Misses"  should be very low (< 2% in most cases)
->"Cache Usage" is the number of cache entries being used
->"Pct SGA"     is the ratio of usage to allocated size for that cache
                          Get         Pct     Scan   Pct      Mod  Final  Pct
Cache                     Requests     Miss Requests  Miss      Req  Usage  SGA
---------------------- ------------ ------ -------- ----- -------- ------ ----
dc_constraints                   9   33.3        0              9    184   93
dc_database_links            2,331    0.0        0              0     20   95
dc_files                       172    0.0        0              0      1    5
dc_free_extents                148   39.9       87   0.0      128    341   23
dc_global_oids                   0              0              0      2    9
dc_histogram_data                0              0              0      0    0
dc_histogram_data_valu           0              0              0      0    0
dc_histogram_defs          831,084    0.0        0             14    702   97
dc_object_ids            1,301,711    0.0        0             29  1,517  100
dc_objects                 139,492    0.1        0             92  2,037  100
dc_outlines                      0              0              0      0    0
dc_profiles                  7,783    0.0        0              0      1   14
dc_rollback_segments        44,335    0.0        0              0    181   99
dc_segments                345,098    0.0        0            195  1,249   99
dc_sequence_grants               0              0              0      0    0
dc_sequences                 5,942    0.0        0          5,939      8   50
dc_synonyms                 61,318    0.0        0              0    205  100
dc_tablespace_quotas             3    0.0        0              3      9   17
dc_tablespaces              19,905    0.0        0              0     43   77
dc_used_extents                122   71.3        0            122    298  100
dc_user_grants              84,772    0.0        0              0     40   98
dc_usernames                80,477    0.0        0              0     26   60
dc_users                   114,086    0.0        0              0     42   86
ifs_acl_cache_entries            0              0              0      0    0
          -------------------------------------------------------------
```

```
Library Cache Activity for DB: dba01  Instance: dba01  Snaps: 2 -4
->"Pct Misses"  should be very low

                    Get       Pct        Pin        Pct               Invali-
Namespace         Requests    Miss     Requests     Miss     Reloads  dations
--------------- ------------ ------ -------------- ------ ---------- --------
BODY                359,086    0.0        359,084    0.0          0        0
CLUSTER               1,575    0.3          1,130    0.9          0        0
INDEX                     0                    0                 0        0
OBJECT                    0                    0                 0        0
PIPE                      0                    0                 0        0
SQL AREA            738,856    3.1     20,928,868    0.2      3,122    1,434
TABLE/PROCEDURE     146,977    0.2      4,040,503    0.0      1,154        0
TRIGGER             374,089    0.0        374,091    0.0         73        0
          -------------------------------------------------------------

SGA Memory Summary for DB: dba01  Instance: dba01  Snaps: 2 -4

SGA regions                     Size in Bytes
------------------------------ ----------------
Database Buffers                  737,280,000
Fixed Size                             76,820
Redo Buffers                          532,480
Variable Size                     379,277,312
                               ----------------
sum                               1,117,166,612
          -------------------------------------------------------------
```

SGA breakdown difference for DB: dba01 Instance: dba01 Snaps: 2 -4

Pool	Name	Begin value	End value	Difference
java pool	free memory	20,684,800	20,684,800	0
java pool	memory in use	286,720	286,720	0
large pool	free memory	25,000,000	25,000,000	0
shared pool	DML locks	3,487,200	3,487,200	0
shared pool	KGFF heap	39,392	39,392	0
shared pool	KGK heap	5,848	5,848	0
shared pool	KQLS heap	2,937,448	2,278,992	-658,456
shared pool	PL/SQL DIANA	2,983,520	336,624	-2,646,896
shared pool	PL/SQL MPCODE	1,697,264	1,318,112	-379,152
shared pool	PLS non-lib hp	2,104	2,104	0
shared pool	PX msg pool	1,100,752	1,100,752	0
shared pool	PX subheap	137,848	137,848	0
shared pool	State objects	6,890,864	6,890,864	0
shared pool	db_block_buffers	12,240,000	12,240,000	0
shared pool	db_handles	3,000,000	3,000,000	0
shared pool	dictionary cache	2,846,608	2,953,016	106,408
shared pool	enqueue_resources	2,109,600	2,109,600	0
shared pool	event statistics per ses	22,932,560	22,932,560	0
shared pool	fixed allocation callbac	3,600	3,600	0
shared pool	free memory	34,661,656	34,407,664	-253,992
shared pool	joxs heap init	4,152	4,152	0
shared pool	ktlbk state objects	3,109,424	3,109,424	0
shared pool	library cache	88,638,712	91,396,248	2,757,536
shared pool	miscellaneous	19,692,288	19,220,056	-472,232
shared pool	processes	4,776,000	4,776,000	0
shared pool	sessions	14,240,384	14,240,384	0
shared pool	sql area	99,182,568	100,743,304	1,560,736
shared pool	table columns	31,144	18,592	-12,552
shared pool	table definiti	25,616	20,104	-5,512
shared pool	transactions	6,480,384	6,480,384	0
shared pool	trigger defini	21,368	29,848	8,480
shared pool	trigger inform	1,120	1,768	648
	db_block_buffers	737,280,000	737,280,000	0
	fixed_sga	76,820	76,820	0
	log_buffer	524,288	524,288	0

```
init.ora Parameters for DB: dba01  Instance: dba01  Snaps: 2 -4
                                                                 End value
Parameter Name                 Begin value                     (if different)
------------------------------ ------------------------------- --------------
_db_handles_cached             0
_trace_files_public            TRUE
audit_file_dest                /logs/oracle/dba01/adump
audit_trail                    FALSE
background_dump_dest           /dba01/ADMIN/CDUMP
compatible                     8.1.7
control_files                  /dba01/ORADATA/ctl/cntrlP
core_dump_dest                 /dba01/ADMIN/CDUMP
db_block_buffers               90000
db_block_lru_latches           24
db_block_size                  8192
db_file_multiblock_read_count  32
db_files                       220
db_name                        dba01
db_writer_processes            8
distributed_transactions       350
global_names                   TRUE
java_pool_size                 20971520
job_queue_processes            3
large_pool_size                25000000, B
log_archive_dest               /dba01/BACKUP/LOG/arc
log_archive_format             %S_%t.arc
log_archive_start              TRUE
log_buffer                     524288
log_checkpoint_interval        250000
max_dump_file_size             512
max_enabled_roles              30
max_rollback_segments          200
nls_date_format                DD-MON-RR
open_cursors                   300
open_links                     16
optimizer_mode                 RULE
parallel_max_servers           25
parallel_min_servers           4
processes                      6000
remote_os_authent              FALSE
remote_os_roles                TRUE
resource_limit                 FALSE
rollback_segments              rbs1_01, rbs1_02, rbs1_03, rbs1_0
session_cached_cursors         90
shared_pool_reserved_size      12582912
shared_pool_size               262144000
sort_area_retained_size        4194304
sort_area_size                 4194304
sql_trace                      FALSE
timed_statistics               TRUE
transactions_per_rollback_seg  50
user_dump_dest                 /dba01/ADMIN/UDUMP/dba01/udump
utl_file_dir                   /cmp/work/UTL_FILE
          -------------------------------------------------------------
End of Report
```

```
transactions_per_rollback_seg 50
user_dump_dest                /dba01/ADMIN/UDUMP/dba01/udump
utl_file_dir                  /cmp/work/UTL_FILE
          -----------------------------------------------------------
End of Report
```

E

Redundant Arrays of Inexpensive Disks Technology (RAID)

System Hardware Configuration

Storage Subsystem Detail

Storage subsystem performance is one of the most important aspects of tuning an Oracle database server for optimal performance. The architecture and design of the storage subsystem must therefore be considered early in the system design process. In performing the storage subsystem design, system requirements such as the required volume of online transaction data, peak transactions per second load, and system availability are transformed into specific storage subsystem design requirements for storage capacity, peak sustainable I/Os per second, and fault tolerance. Values for design parameters in the selected technology are then chosen to meet these specific requirements.

Modern storage systems offer great flexibility in meeting a wide range of design criteria; technologies such as striping, mirroring, and other fault-tolerant RAID configurations provide the ability to meet these design requirements. Matching the right technology with application I/O characteristics is key to achieving the promised performance and fault tolerance levels; conversely, using the wrong technology for a specific I/O characteristic can lead to I/O bottlenecks and degraded response times. In this section, key parameters in the design of the storage subsystem are described, as well as how they relate to the performance of an Oracle database.

Storage Subsystem Design Parameters and Oracle

When designing a storage subsystem, the available design parameters are weighed against each other until a design solution is achieved that meets or exceeds all design requirements. In the context of an Oracle database server, certain measures are available to specify these requirements. These measures can be categorized under performance, availability, and cost.

Performance

- Random read performance: Important for Oracle indexed or hash-based queries and rollback segment reads
- Random write performance: Important for Oracle DBWn writes; heavy in an OLTP environment, light in a data warehouse
- Sequential read performance: Backups, Oracle full table scans, index creations, parallel queries, temporary segment reads, and recovery from archived redo log files
- Sequential write performance: Oracle LGWR writes, temporary segment writes, direct-path loader writes, tablespace creations
- Impact of concurrency

Storage Subsystem Design Parameters and Oracle (continued)

Availability

- Outage frequency: Expected number of occurrences of a possible outage per unit of time specified in mean time to failure (MTTF)
- Outage duration: The mean time to repair (MTTR) for a given outage event
- Performance degradation during outage: Whether a disk configuration provides service during a fault, and if so, at what level

Cost

- Acquisition cost: The cost of purchasing, installing, and configuring the storage subsystem
- Operational cost: The cost of running and maintaining the system to meet the system availability and service level requirements

The Redundant Arrays of Inexpensive Disks (RAID) technologies have been developed for nonmainframe, open system solutions. RAID provides low-cost fault tolerance and improved performance. Several levels of RAID are available and may be mixed within one storage subsystem design. Each level of RAID can be categorized against the measures listed above and its impact on Oracle database performance. Some levels of RAID configurations have been available for many years under different names. Key parameters when configuring a RAID system are:

- Array size: The number of drives in the array
- Disk size: The size of each disk
- Stripe size: The size of an I/O chunk, written to or read from a contiguous location on a disk (Striping allows data files to be interleaved and spread across the disks in an array in an attempt to parallelize file I/O.)

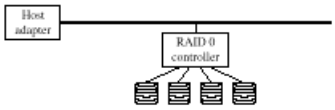
Storage Subsystem Design Parameters and Oracle (continued)

Cost (continued)

The throughput of a storage subsystem, expressed in I/Os per second, determines how many transactions can be processed by the subsystem before queuing delays begin to occur. If the I/Os-per-second requirement of the application is known, broken down into reads per transaction, writes per transaction, and transactions per second, you can determine whether a particular RAID configuration will support the transaction rate applied by an application. To calculate throughput, or total sustainable I/Os-per-second load that a RAID array can support, simply multiply the number of drives in the array times the sustainable I/Os per second of one drive, currently in the approximate range of 35 to 50 I/Os per second. Different RAID configurations, however, add to the I/O load on a disk array applied by the application in order to provide the fault tolerance function. Once the total I/O load on the array is known, the number of drives required to support the load can be found. Simply divide the total I/Os per second load by the random I/O throughput rating for the selected drive.

The sections below briefly describe each RAID level and some of its characteristics. For each level, an equation is provided to calculate the total I/Os per second load on a RAID array, to illustrate the impact of the RAID configuration on the array's capacity. Also provided is an equation for calculating the size of the disk drives required for an array.

RAID Level 0, Nonredundant Striping



RAID 0 refers to simple data striping of multiple disks into a single logical volume, and has no fault tolerance. When properly configured, it provides excellent response times for high concurrency random I/O and excellent throughput for low concurrency sequential I/O. Selection of the array and stripe sizes requires careful consideration in order to achieve the promised throughput. For RAID 0, the total I/Os-per-second load generated against the array is calculated directly from the application load, because there is no fault tolerance in this configuration:

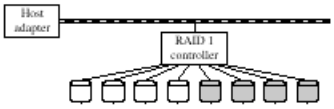
```
total I/O per second load on array = (reads/transaction +
writes/ transaction) * transactions/second
```

The size of each drive in the array can be calculated from the online volume requirements as follows:

```
drive size = [total space required by application / number
of drives in array] Rounded up to next drive size.
```

- Below is a summary of RAID 0 characteristics:
- Random read performance: Excellent under all concurrency levels if each I/O request fits within a single striping segment
- Random write performance: Same as random read performance
- Sequential read performance: Excellent with fine-grained striping at low concurrency levels
- Sequential write performance: Same as sequential read performance
- Outage frequency: Poor; any single disk failure will cause application outage
- Outage duration: Poor; the duration of a RAID 0 outage is the time required to detect the failure, replace the disk drive, and perform Oracle media recovery
- Performance degradation during outage: Poor; any disk failure causes all applications requiring use of the array to crash
- Acquisition cost: Excellent, because there is no redundancy; you buy only enough for storage and I/Os per second requirements
- Operational cost: Fair to poor; frequent media recoveries increase operational costs and may outweigh the acquisition cost advantage

RAID Level 1, Mirroring



RAID Level 1, or disk mirroring, provides the best fault tolerance of any of the RAID configurations. Each disk drive is backed up by an exact copy of itself on an identical drive. A storage subsystem of mirrored drives can continue at full performance with a multiple disk failure as long as no two drives in a mirrored pair have failed. The total I/Os per load applied to a mirrored pair is calculated as follows:

```
total I/O per second load on array = (reads/transaction +
2*writes/ transaction) * transactions/second
```

Note the two multiplier of the writes/transaction factor. This is due to the fact that each write request by an application to a mirrored pair actually results in two writes, one to the primary disk and one to the backup disk. The size of the drive required is:

```
drive size = [total space required by application / number
of drives in array/2] Rounded up to next drive size.
```

In the simplest RAID 1 configuration, the number of drives in the array is two: the primary drive and its backup. The definition of RAID 1, however, includes the ability to expand the array in units of two drives to achieve a striped and mirrored configuration. Striping occurs in an array of four or more disks. Some industry literature (for example, Millsap,1996) refers to striped and mirrored configurations as RAID 0 + 1. The Compaq hardware used as an example configuration in this document supports both configurations. Compaq uses only the RAID 1 term to describe all 100% mirrored configurations in arrays of even-numbered disks. Because the performance of a simple two-drive RAID 1 pair is somewhat different from a striped and mirrored array, the figures for striped and mirrored are presented separately under the RAID 0 + 1 section.

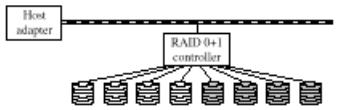
Below is a summary of characteristics of the two-disk array RAID configuration:

- Random read performance: Good; if the implementation uses read-optimized RAID 1 controllers, which read from the drive with the smallest I/O setup cost, then slightly better than an independent disk
- Random write performance: Good (Application write requests are multiplied by two, because the data must be written to two disks. Thus, some of the I/Os-per-second capacity of the two drives is used up by the mirroring function.)

RAID Level 1, Mirroring (continued)

- Sequential read performance: Fair; throughput is limited to the speed of one disk
- Sequential write performance: Fair; same factors as are influencing the random write performance
- Outage frequency: Excellent
- Outage duration: Excellent; for “hot swapable” drives, no application outage is encountered by a single failure
- Performance degradation during outage: Excellent; there is no degradation during a disk outage (After replacing of the failed drive, the resilvering operation that takes place when the failed disk is replaced will consume some of the available I/Os per second capacity.)
- Acquisition cost: Poor; each RAID 1 pair requires two drives to achieve the storage capacity of one
- Operational cost: Fair; increased complexity of the configuration leads to higher training costs and costs to develop custom software to integrate the mirroring procedures into scheduled maintenance operations

RAID Level 0+1, Striping and Mirroring

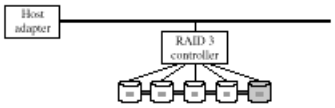


As noted in the previous section, the striped and mirrored configuration is an expansion of the RAID 1 configuration from a simple mirrored pair to an array of even-numbered drives. This configuration offers the performance benefits of RAID 0 striping and the fault tolerance of simple RAID 1 mirroring. The striped and mirrored configuration is especially valuable for Oracle data files holding files with high write rates, such as table data files and online and archived redo log files. Unfortunately, it also presents the high costs of simple RAID 1. The equations for the total I/Os per second and disk drive size calculations for RAID 0 + 1 are identical to those presented for RAID 1 above. The Compaq SMART array controller used in the example configuration supports RAID 0 + 1 (RAID 1 in Compaq terminology) in arrays up to 14 drives, providing the effective storage of 7 drives.

Below is a summary of characteristics of RAID 0 + 1 storage arrays:

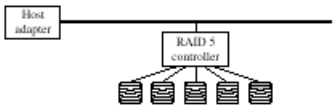
- Random read performance: Excellent under all concurrency levels if each I/O requests fits within a single striping segment (Using a stripe size that is too small can cause dramatic performance breakdown at high concurrency levels.)
- Random write performance: Good (Application write requests are multiplied by two because the data must be written to two disks. Thus, some of the I/Os-per-second capacity of the two drives is used up by the mirroring function.)
- Sequential read performance: Excellent under all concurrency levels if each I/O request fits within a single striping segment
- Sequential write performance: Good
- Outage frequency: Excellent; same as RAID 1
- Outage duration: Excellent; same as RAID 1
- Performance degradation during outage: Excellent, there is no degradation during a disk outage (The resilvering operation that takes place when the failed disk is replaced will consume a significant amount of the available I/Os-per-second capacity.)
- Acquisition cost: Poor; same as RAID 1
- Operational cost: Fair; same as RAID 1

RAID Level 3, Bit Interleaved Parity



In the RAID 3 configuration, disks are organized into arrays in which one disk is dedicated to storage of parity data for the other drives in the array. The stripe size in RAID 3 is 1 bit. This enables recovery time to be minimized, because data can be reconstructed with a simple exclusive-OR operation. However, using a stripe size of 1 bit reduces I/O performance. RAID 3 is not recommended for storing any Oracle database files. Also, RAID 3 is not supported by the Compaq SMART array controllers.

RAID Level 5, Block-Interleaved with Distributed Parity



RAID 5 is similar to RAID 3, except that RAID 5 striping segment sizes are configurable, and RAID 5 distributes parity across all the disks in an array. A RAID 5 striping segment contains either data or parity.

Battery-backed cache greatly reduces the impact of this overhead for write calls, but its effectiveness is implementation-dependent. Large write-intensive batch jobs generally fill the cache quickly, reducing its ability to offset the write-performance penalty inherent in the RAID 5 definition.

The total I/Os per second load applied to a RAID 5 array is calculated as follows:

```
total I/O per second load on array = (reads/transaction +
4*writes/ transaction) * transactions/second
```

The writes/transaction figure is multiplied by four because the parity data must be written in a six-step process:

1. Read the data drive containing the old value of the data to be overwritten. This requires one I/O.
2. Read the parity drive. This requires one I/O.
3. Subtract the contribution of the old data from the parity value.
4. Add the contribution of the new data to the parity value.
5. Write the new value of the parity requiring one I/O.
6. Write the new data value to the data drive. This requires one I/O.

Summing up all I/Os in this process yields four I/Os required for each write requested by the application. This is the main reason that RAID 5 is not recommended for storing files with a high I/O performance requirement; the 4 multiplier reduces the effective I/Os-per-second capacity of the array.

The size of the drive required is:

```
drive size = [total space required by application /(total
number drives - number of arrays)] Rounded up to next
drive size.
```

RAID Level 5, Block-Interleaved with Distributed Parity (continued)

Note that the figure "number of arrays" is used to account for the space of one drive per array consumed by the parity data. If it is necessary to exceed the maximum recommended array size to meet the I/Os-per-second performance requirement, then multiple arrays are required.

Below is a summary of the characteristics of RAID 5 storage arrays:

- Random read performance: Excellent under all concurrency levels if each I/O requests fits within a single striping segment (Using a stripe size that is too small can cause dramatic performance breakdown at high concurrency levels.)
- Random write performance: Poor; worst at high concurrency levels (The read-modify-write cycle requirement of RAID 5 parity implementation reduces the effective throughput or I/Os-per-second capacity of the array, especially for heavy write I/O files. It should be noted, however, that under light load, response time is not degraded from that provided by a faster array configuration such as RAID 0. This is due to the asynchronous write capabilities provided by most array controllers. With asynchronous write, the application does not have to wait until the storage subsystem has completed the read-modify-write cycle before continuing. Instead, the controller buffers the data in battery-backed RAM, signals the application that the write has completed, then completes the write to disk. (This buffering does nothing to increase throughput, however.)
- Sequential read performance: Excellent under high concurrency levels if each I/O request fits within a single striping segment; also excellent with fine grain striping under low concurrency levels
- Sequential write performance: Fair for low concurrency levels, poor for high concurrency levels (See random write performance.)
- Outage frequency: Good; can withstand the loss of any single disk in a given array without incurring an application outage (Multiple simultaneous disk failures causes application outage. The possibility of multiple simultaneous failures increase as the size of the array increases.)
- Outage duration: Good; a single disk failure causes no application outage
- Performance degradation during outage: Fair; there is no degradation for reads and writes to or from surviving drives in the array (Reads and writes to a failed drive incur a high performance penalty, requiring data from all surviving drives in the array to be read. Reconstruction of the failed drive's data also degrades performance.)
- Acquisition cost: Fair (If storage capacity were the only factor, the cost would be g/(g-1) times the cost of the equivalent RAID 0 capacity, where g is the number of disks in the array. However, when factoring I/Os-per-second performance requirement, the cost can meet or exceed the cost of a RAID 0 + 1 implementation.)
- Operational cost: Fair (Training is required to configure striped disk arrays for optimal performance.)

Ranking of RAID Levels Against Oracle File Types

The following table provides relative rankings for RAID configurations for specific Oracle file types. The rankings range from 1 (Best) to 5 (Worst). Adapted from Millsap,1996, page 13.

Query	None	0	1	0+1	3	5
Control file performance	2	1	2	1	5	3
Redo log file performance	4	1	5	1	2	3
System tablespace performance	2	1	2	1	5	3
Sort segment performance	4	1	5	1	2	3
Rollback segment performance	2	1	2	1	5	5
Indexed read-only datafiles	2	1	2	1	5	1
Sequential read-only datafiles	4	1	5	1	2	3
DBWn intensive data files	1	1	2	1	5	5
Direct load-intensive data files	4	5	1	1	2	2
Data protection	4	5	1	1	2	2
Acquisition and operating costs	1	1	5	5	3	3