# Oracle Database 10*g*: SQL Tuning Workshop

**Student Guide • Volume 2**

D17265GC11

Edition 1.1

March 2007

D49769

## Author

Priya Vennapusa

## Technical Contributors and Reviewers

Andrew Brannigan
Cecelia Gervasio
Chika Izumi
Connie Green
Dairy Chan
Donna Keesling
Graham Wood
Harald Van Breederode
Helen Robertson
Janet Stern
Jean Francois Verrier
Joel Goodman
Lata Shivaprasad
Lawrence Hopper
Lillian Hobbs
Marcelo Manzano
Martin Jensen
Mughees Minhas
Ric Van Dyke
Robert Bungenstock
Russell Bolton
Dr. Sabine Teuber
Stefan Lindblad

## Editor

Raj Kumar

## Graphic Designer

Rajiv Chandrabhanu

## Publisher

Jobi Varghese

**Disclaimer**

This document contains proprietary information and is protected by copyright and other intellectual property laws. You may copy and print this document solely for your own use in an Oracle training course. The document may not be modified or altered in any way. Except where your use constitutes "fair use" under copyright law, you may not use, share, download, upload, copy, print, display, perform, reproduce, publish, license, post, transmit, or distribute this document in whole or in part without the express authorization of Oracle.

The information contained in this document is subject to change without notice. If you find any problems in the document, please report them in writing to: Oracle University, 500 Oracle Parkway, Redwood Shores, California 94065 USA. This document is not warranted to be error-free.

**Restricted Rights Notice**

If this documentation is delivered to the United States Government or anyone using the documentation on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS
The U.S. Government's rights to use, modify, reproduce, release, perform, display, or disclose these training materials are restricted by the terms of the applicable Oracle license agreement and/or the applicable U.S. Government contract.

**Trademark Notice**

Oracle, JD Edwards, PeopleSoft, and Siebel are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

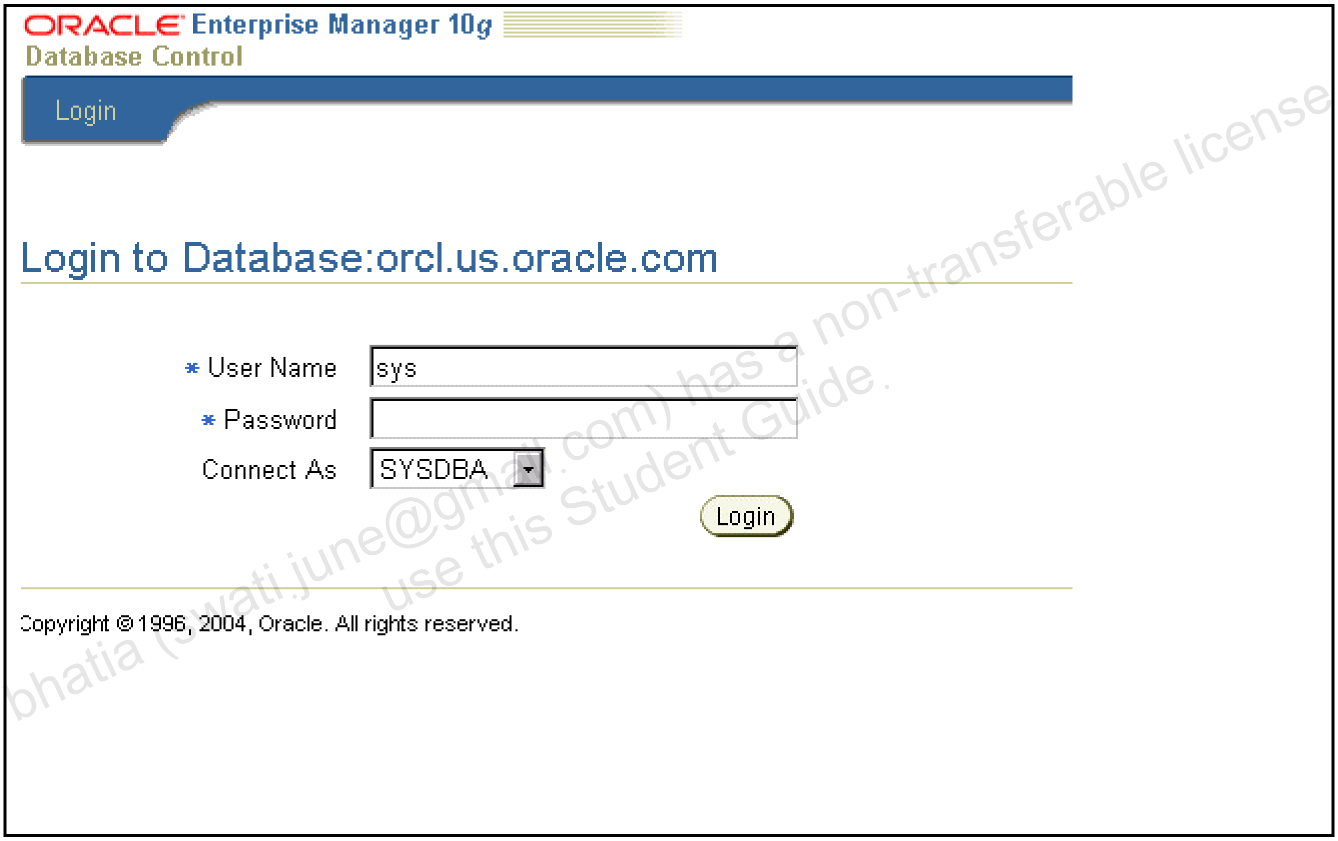
# Contents

# Appendix A: Practices

## Practice 1

1. Launch a browser and enter the URL `http:// machine_name:1158/em`, where *`machine_name`* stands for the IP address of your local machine. Log in to Enterprise Manager as `sys` with the password `SYSDBA`, and then click **Login**. If you are logging in for the first time, a license agreement may appear. Click **I Agree** to proceed.

**Note:** You may also see an Adobe dialog box, in which you must accept an agreement before graphics can appear properly.



Login to Database:orcl.us.oracle.com

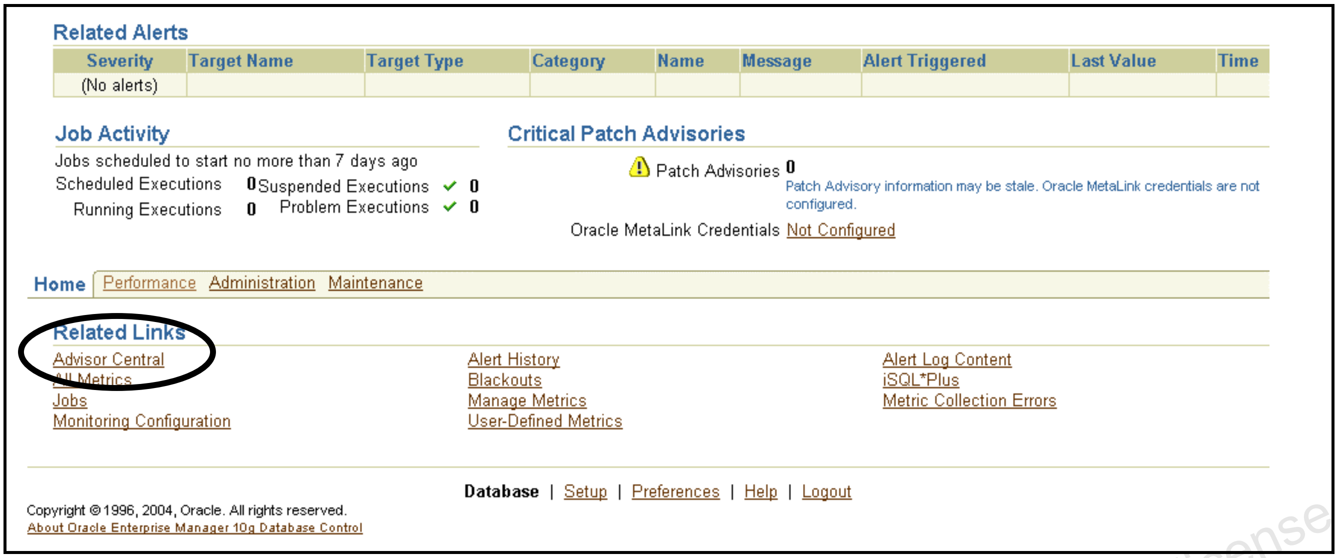
* User Name: sys

* Password:

Connect As: SYSDBA

Copyright © 1996, 2004, Oracle. All rights reserved.

2. Scroll to the bottom of the home page and click **Advisor Central** under **Related Links**.



Related Alerts

| Severity | Target Name | Target Type | Category | Name | Message | Alert Triggered | Last Value | Time |
|---|---|---|---|---|---|---|---|---|
| (No alerts) | | | | | | | | |

Job Activity

Jobs scheduled to start no more than 7 days ago

| | | | |
|---|---|---|---|
| Scheduled Executions | 0 | Suspended Executions | ✓ 0 |
| Running Executions | 0 | Problem Executions | ✓ 0 |

Critical Patch Advisories

⚠ Patch Advisories 0

Patch Advisory information may be stale. Oracle MetaLink credentials are not configured.

Oracle MetaLink Credentials Not Configured

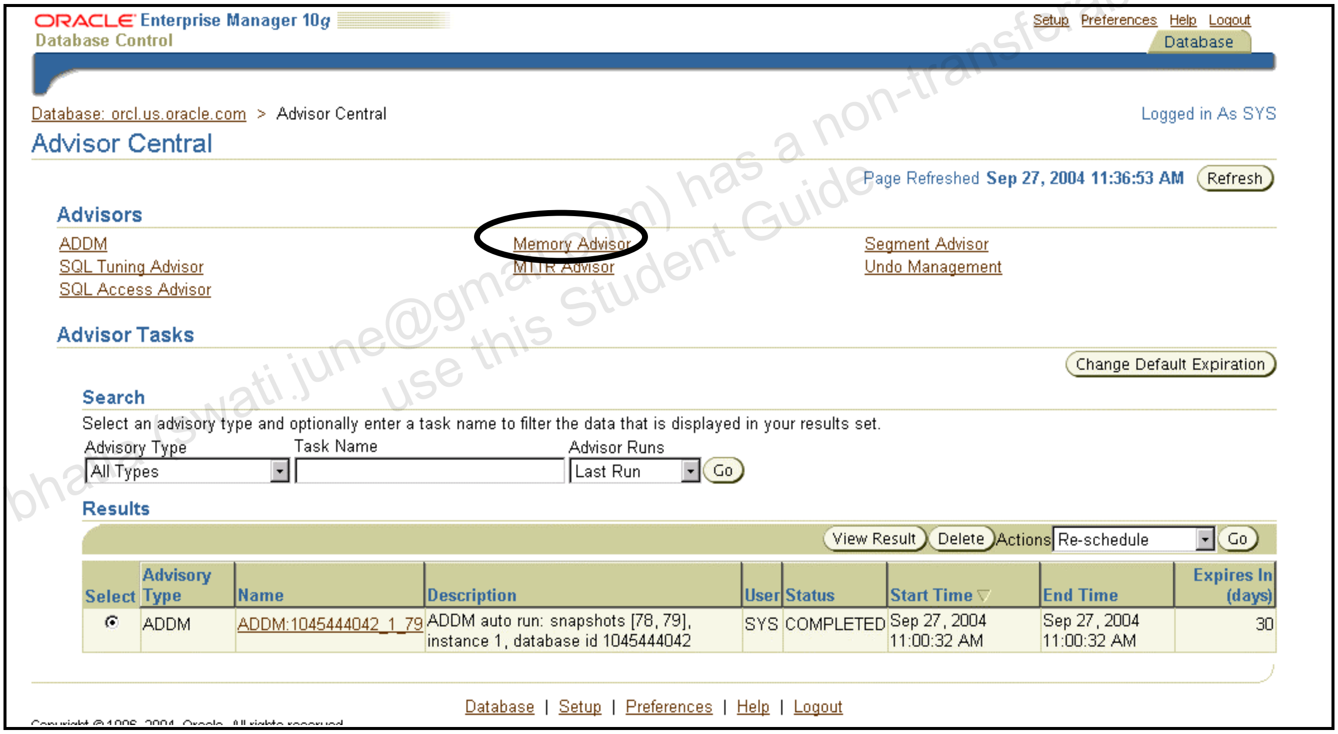
Related Links

| | | |
|---|---|---|
| Advisor Central | Alert History | Alert Log Content |
| All Metrics | Blackouts | iSQL*Plus |
| Jobs | Manage Metrics | Metric Collection Errors |
| Monitoring Configuration | User-Defined Metrics | |

Copyright © 1996, 2004, Oracle. All rights reserved.

About Oracle Enterprise Manager 10g Database Control

3. Click the **Memory Advisor** link under **Advisors**.



ORACLE Enterprise Manager 10g
Database Control
Setup Preferences Help Logout
Database

Database: orcl.us.oracle.com > Advisor Central

Logged in As SYS

Advisor Central

Page Refreshed **Sep 27, 2004 11:36:53 AM** (Refresh)

Advisors

| | | |
|---|---|---|
| ADDM | Memory Advisor | Segment Advisor |
| SQL Tuning Advisor | MTTR Advisor | Undo Management |
| SQL Access Advisor | | |

Advisor Tasks

Change Default Expiration

Search

Select an advisory type and optionally enter a task name to filter the data that is displayed in your results set.

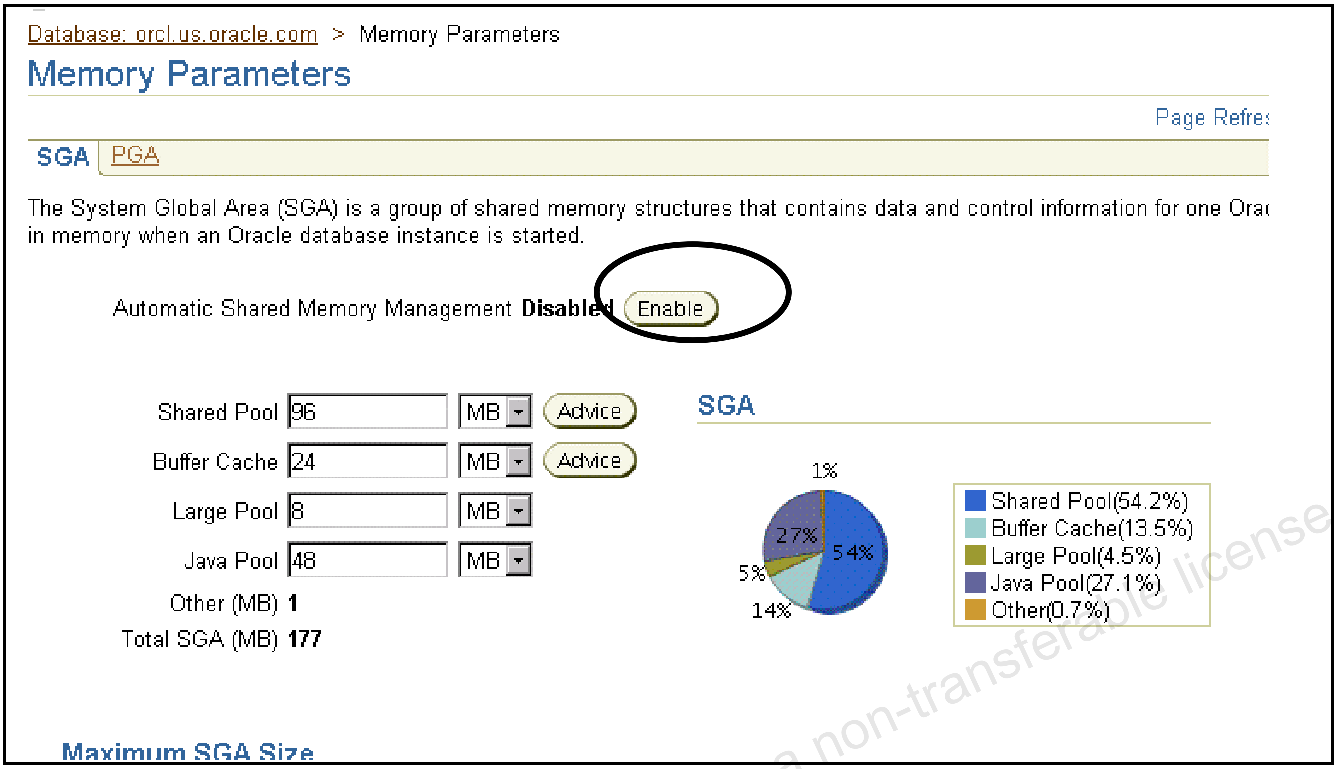
Advisory Type: All Types
Task Name:
Advisor Runs: Last Run (Go)

Results

| Select | Advisory Type | Name | Description | User | Status | Start Time ▽ | End Time | Expires In (days) |
|---|---|---|---|---|---|---|---|---|
| ● | ADDM | ADDM:1045444042_1_79 | ADDM auto run: snapshots [78, 79], instance 1, database id 1045444042 | SYS | COMPLETED | Sep 27, 2004 11:00:32 AM | Sep 27, 2004 11:00:32 AM | 30 |

Copyright © 1996, 2004, Oracle. All rights reserved.

4. Click the **Enable** button for **Automatic Shared Memory Management**.



Database: orcl.us.oracle.com > Memory Parameters

Memory Parameters

Page Refres

SGA | PGA

The System Global Area (SGA) is a group of shared memory structures that contains data and control information for one Orac
in memory when an Oracle database instance is started.

Automatic Shared Memory Management **Disabled** (Enable)

Shared Pool 96 MB (Advice)

Buffer Cache 24 MB (Advice)

Large Pool 8 MB

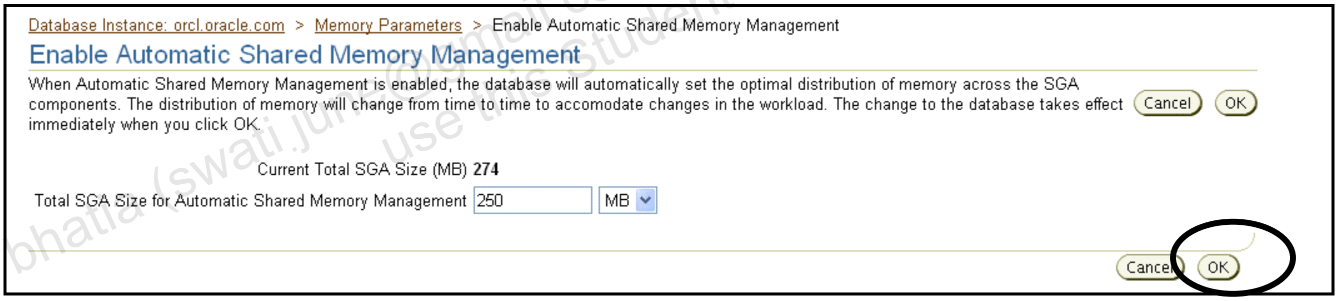
Java Pool 48 MB

Other (MB) **1**

Total SGA (MB) **177**

Maximum SGA Size

5. Set Total SGA for Shared Memory Management to 250 MB. Click **OK** to enable **Automatic Shared Memory Management**.



Database Instance: orcl.oracle.com > Memory Parameters > Enable Automatic Shared Memory Management

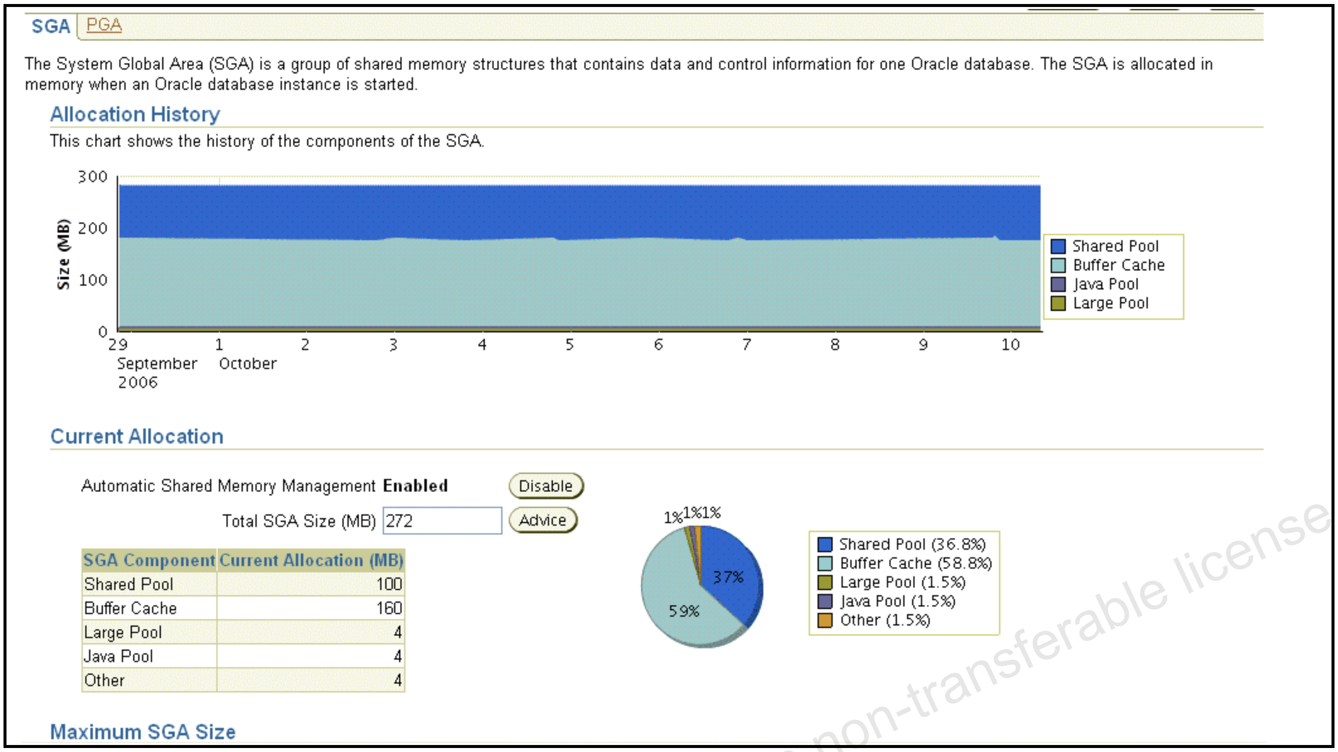
Enable Automatic Shared Memory Management

When Automatic Shared Memory Management is enabled, the database will automatically set the optimal distribution of memory across the SGA components. The distribution of memory will change from time to time to accomodate changes in the workload. The change to the database takes effect immediately when you click OK.

Current Total SGA Size (MB) **274**

Total SGA Size for Automatic Shared Memory Management 250 MB

6. The Oracle database will now automatically adjust the settings for the various pools and caches according to the requirements of the workload.



The System Global Area (SGA) is a group of shared memory structures that contains data and control information for one Oracle database. The SGA is allocated in memory when an Oracle database instance is started.

## Allocation History

This chart shows the history of the components of the SGA.

## Current Allocation

Automatic Shared Memory Management **Enabled**

Total SGA Size (MB) 272

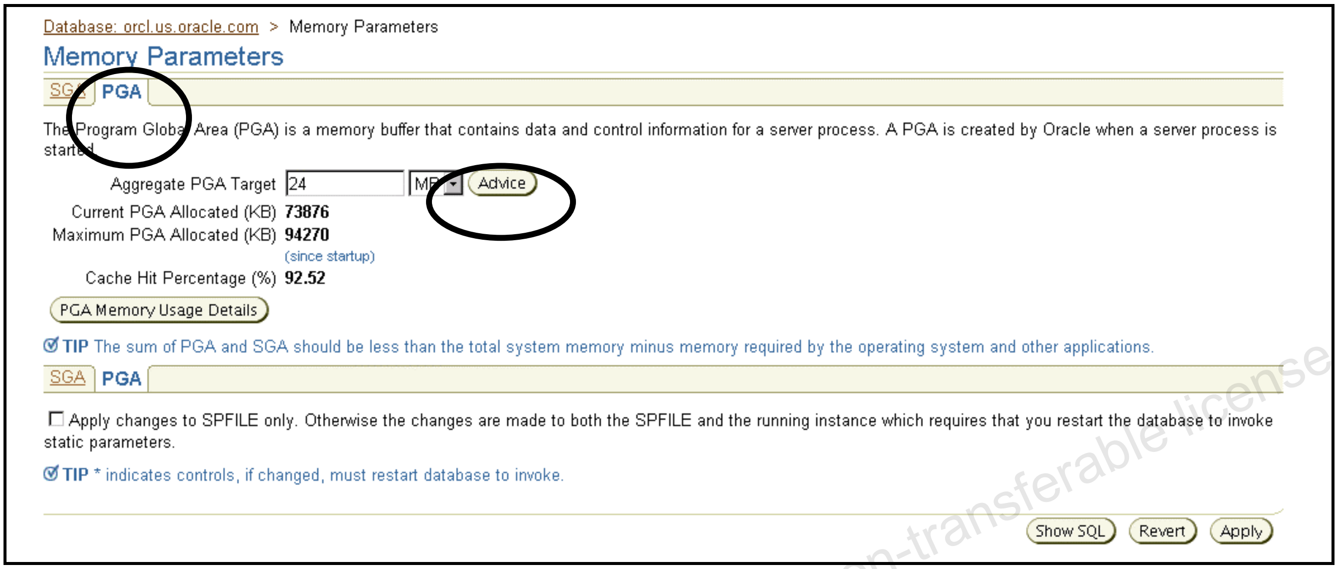
| SGA Component | Current Allocation (MB) |
|---|---|
| Shared Pool | 100 |
| Buffer Cache | 160 |
| Large Pool | 4 |
| Java Pool | 4 |
| Other | 4 |

## Maximum SGA Size

### Using the PGA Advisor

To allocate memory associated with the PGA, perform the following steps:

1. Click the **PGA** tab.



Database: orcl.us.oracle.com > Memory Parameters

Memory Parameters

The Program Global Area (PGA) is a memory buffer that contains data and control information for a server process. A PGA is created by Oracle when a server process is started.

Aggregate PGA Target 24 MB Advice

Current PGA Allocated (KB) **73876**

Maximum PGA Allocated (KB) **94270**
(since startup)

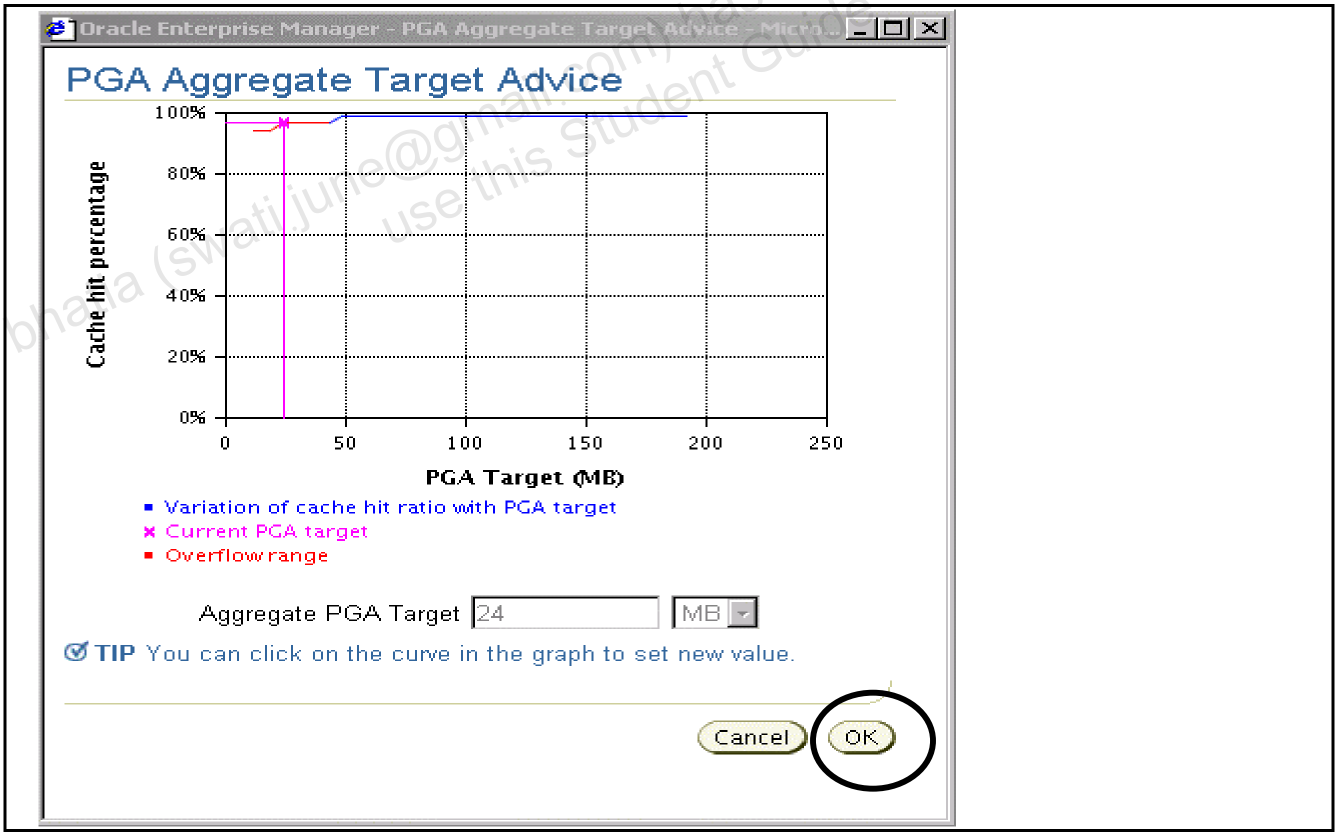
Cache Hit Percentage (%) **92.52**

TIP The sum of PGA and SGA should be less than the total system memory minus memory required by the operating system and other applications.

□ Apply changes to SPFILE only. Otherwise the changes are made to both the SPFILE and the running instance which requires that you restart the database to invoke static parameters.

TIP * indicates controls, if changed, must restart database to invoke.

2. Click the **Advice** button.



Oracle Enterprise Manager - PGA Aggregate Target Advice - Micro...

PGA Aggregate Target Advice

Cache hit percentage
100%
80%
60%
40%
20%
0%
0
50
100
150
200
250
PGA Target (MB)
■ Variation of cache hit ratio with PGA target
✖ Current PGA target
■ Overflow range

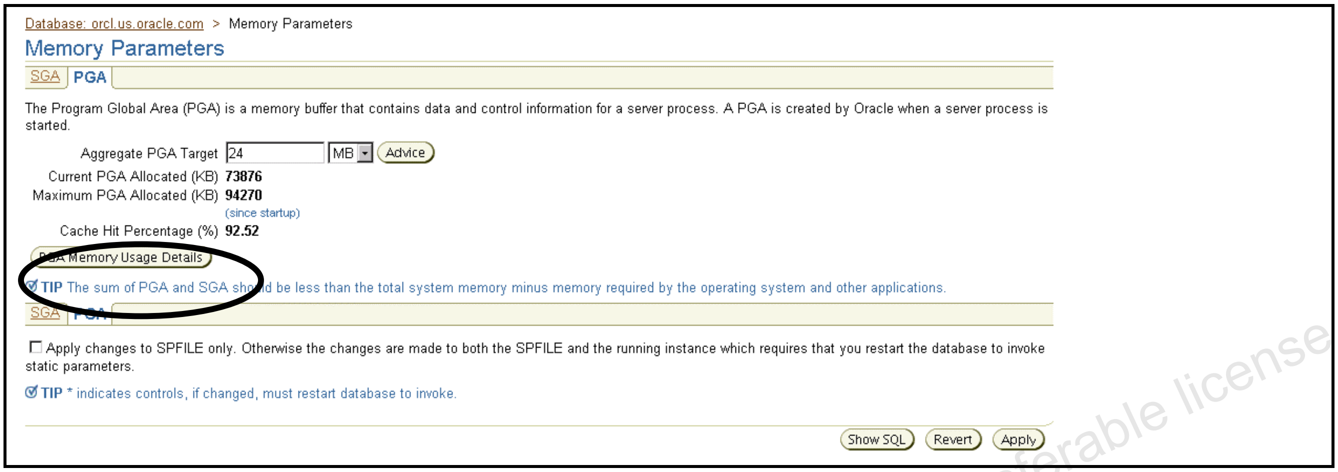
Aggregate PGA Target 24 MB

TIP You can click on the curve in the graph to set new value.

3. The PGA Aggregate Target Advice graph shows the frequency with which data is found in the cache so that you do not have to access disk memory. In this case, note that the current PGA aggregate size is set to approximately 24 MB, and over 88% of all the requested services are obtained from memory. This also shows the overflow range, which starts around

12 MB. At 12 MB, the PGA requests use up the cache to around 90%. The PGA aggregate size implies that (based on current workloads and the number of sessions in the database) no more than 24 MB should be allocated for all PGA in this database. Click the **OK** button.

4. Click the **PGA Memory Usage Details** button.



Database: orcl.us.oracle.com > Memory Parameters

Memory Parameters

SGA | PGA

The Program Global Area (PGA) is a memory buffer that contains data and control information for a server process. A PGA is created by Oracle when a server process is started.

Aggregate PGA Target 24 MB Advice

Current PGA Allocated (KB) **73876**

Maximum PGA Allocated (KB) **94270**

(since startup)

Cache Hit Percentage (%) **92.52**

PGA Memory Usage Details

✓ **TIP** The sum of PGA and SGA should be less than the total system memory minus memory required by the operating system and other applications.

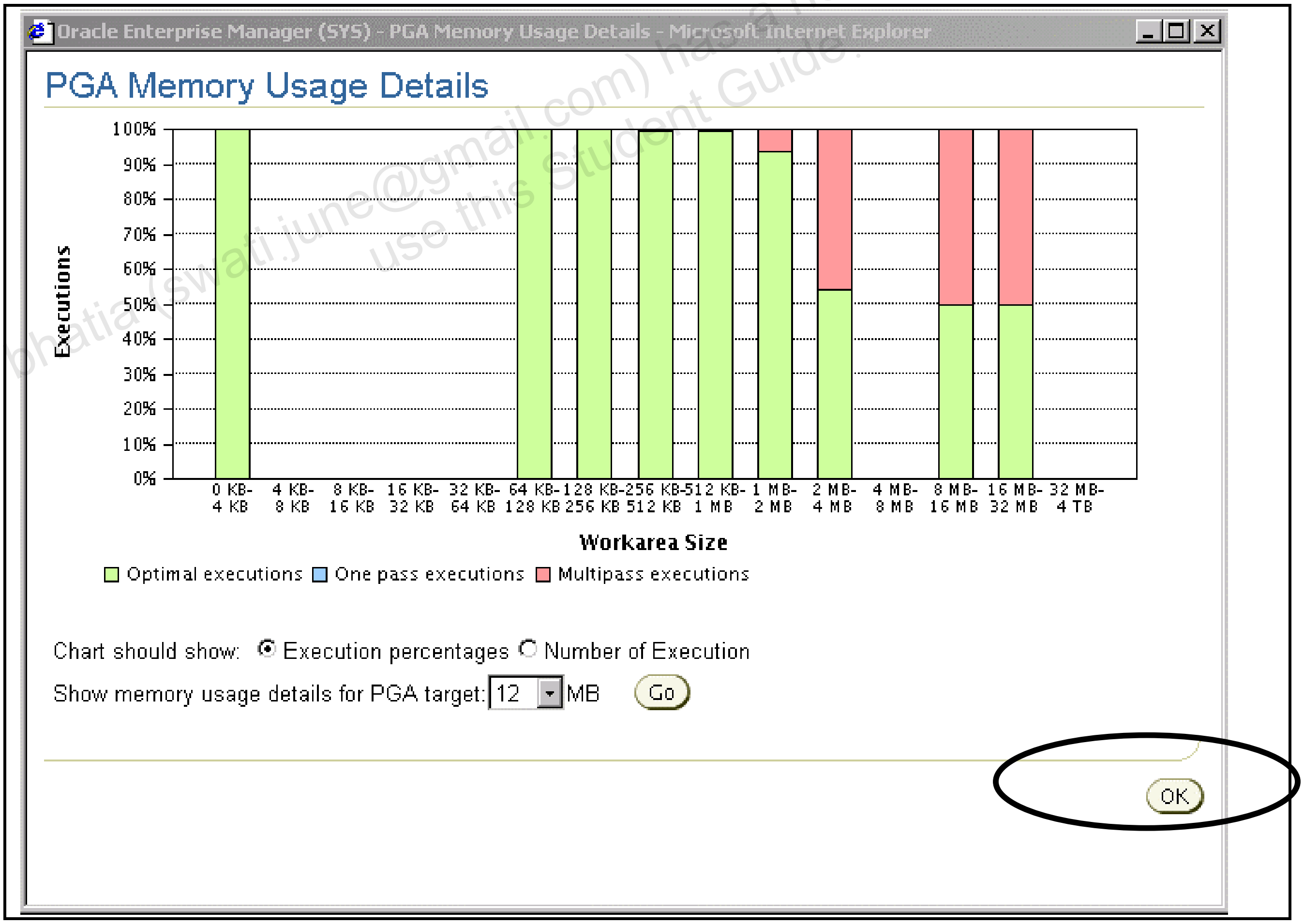
SGA | PGA

□ Apply changes to SPFILE only. Otherwise the changes are made to both the SPFILE and the running instance which requires that you restart the database to invoke static parameters.

✓ **TIP** * indicates controls, if changed, must restart database to invoke.

Show SQL | Revert | Apply

5. This graph shows the usage details in memory size requests and execution percentages for various PGA memory requests. Click **OK** to proceed.



PGA Memory Usage Details

Chart should show: ● Execution percentages ○ Number of Execution

Show memory usage details for PGA target: 12 MB Go

OK

## Practice 3

1. Open a terminal window. Set the current directory to **`labs`**. Log in to SQL*Plus as `SH` with password `SH`.

   This lab demonstrates cursor sharing. Use **`flush.sql`** to flush the shared pool. View and run the scripts **`lab_03_01.sql`** and **`lab_03_01b.sql`**. These execute the same statement with different values in the predicate.

2. Then use **`lab_03_02.sql`** to select from `V$SQL_AREA`. What do you see? Why?

## Practice 6

1. Open a terminal window. Set the current directory to **labs**. Start SQL*Plus. Log in to SH schema as SH with the password SH. Then check the existence and column structure of the CUSTOMERS table. View the structure of the PLAN_TABLE table. Also list the existing indexes on the CUSTOMERS table by using the **li.sql** script. Enable AUTOTRACE TRACEONLY EXPLAIN to suppress statement output and produce execution plans. Check the settings with SHOW AUTOTRACE. Disable AUTOTRACE.
2. Explain the SQL statement that is provided in **lab_06_02.sql**.
   a. Use the **rp.sql** script to query the PLAN_TABLE table in a sophisticated manner.

   b. Enable SQL*Plus AUTOTRACE to display execution plans (**attox.sql)** and run the query again without the EXPLAIN PLAN command by using **lab_06_02b.sql**.

   c. If you have some time left, enter your own SQL statements and experiment with EXPLAIN PLAN and AUTOTRACE. Remember to disable AUTOTRACE when you are finished.

3. Now retrieve information from V$SQLAREA. Execute the script **lab_06_03.sql**.

   Obtain **sqlid** for the statement that you executed. You can use sqlid.sql to help you do this. Using the sqlid, obtain the execution plan from the V$SQLAREA view. You can use the rpsqlarea.sql script to help do this. Remember to substitute your sqlid in the query.
4. Take a look at some SQL from the AWR.
   a. Query V$SQL_TEXT to obtain the sqlid for the statement containing the text "REPORT". You can run the script **lab_06_04a.sql** to execute this statement. First run **flush.sql** to flush the shared pool.

   b. Using sqlid, verify that this statement has been captured in the DBA_HIST_SQLTEXT dictionary view.

   c. If the query does not return rows, then it indicates that the statement has not yet been loaded in the AWR.You can take a manual AWR snapshot rather than wait for the next snapshot (which occurs every hour).

Then check to see if it has been captured in DBA_HIST_SQLTEXT using the same query as the previous step.

   d. Use the DBMS_XPLAN.DISPLAY_AWR () function to retrieve the execution plan.

## Practice 7

1. Open a terminal window. Set the current directory to **labs**. Start SQL*Plus. Log in to SH schema as SH with the password SH. Verify the existence of the statistics-gathering job by querying from DBA_SCHEDULER_JOBS.
2. Create a MY_CUST table that is identical to the customers table. You can use lab_07_02.sql to do this if you like.

   List the indexes on the table using the **li.sql** script.

3. Query USER_TABLES to verify the existence of statistics. You can use the **tabstats.sql** script to do this.
4. Run the query in lab_07_04.sql on the table, and then view the execution plan using **attox.sql**. This script sets AUTOTRACE to TRACE ONLY EXPLAIN.
5. Disable sqltrace by using **atoff.sql**. Create an index on the CUST_ID column by using the **ci.sql** script.
6. Run the query again and view the execution plan using sqltrace.
7. How does the optimizer know to use the index? Disable AUTOTRACE first by using **atoff.sql**.

   **Hint:** Query user indexes and USER_TABLES by using **tabstats.sql** and **indstats.sql** to get the answer.

8. Drop the index that you created. Then verify the index statistics again. What do you see?
9. Identify the date of the last analysis, and the sample size for all tables in your schema. You can use the **schemastats.sql** script to help you do this.

   Identify the types of histograms for all columns in your schema. Try running the **stats.sql** script. Do you see any difference?

   **Hint:** Query **user_tab_col_statistics** or use the **colhist.sql** script.

10. Now consider bind-variable peeking. First run **flush.sql** to flush the shared pool. Then run the script **lab_07_10.sql** that creates the NEW_CUST table and populates it with skewed data. It also creates an index and a histogram on the skew data column cust_id. After this script is run, the CUST_ID column has 1,000 rows with a value of 1, one row with a value of 2, and one row with a value of 3.

    Now run **lab_07_10a.sql** and use **rp.sql** to get the execution plan.

    You see that this is a full table scan. Now try running **lab_07_10b.sql**. Is the index used?

## Practice 8

1. Open a terminal window. Set the current directory to **labs**. Start SQL*Plus. Log in to SH schema as SH with the password SH. Make sure that AUTOTRACE is disabled by using **atoff.sql**.

2. Provide an identifier for the trace file to help you locate it. Drop all indexes on the CUSTOMERS table. Enable SQL Trace. Analyze the SQL statement in **lab_08_01.sql** by using SQL Trace and TKPROF.

3. Now create an index on the CUST_CITY column by using **ci.sql**, and then run the same query again.

4. Disable tracing.

5. Determine the location of the trace files by using the SHOW PARAMETER DUMP command and making note of the UDUMP destination.

6. Exit your session.

7. Change directory to the UDUMP destination.

8. Locate your file by the file identifier that you gave at step 2. Look for a file called **orcl_ora_xxxxx_filename** (for example, orcl_ora_123456_User12.trc).

9. View the difference in execution plans and statistics of the SQL statement with and without an index. You can use gedit to do this. Change back to your home directory and then to the **labs** directory.

10. Take a look at DBMS_MONITOR. Start two sessions, one connected as SYS/ORACLE as SYSDBA and the other connected as SH.

11. From the SYSDBA session, determine the session ID (**sid**) and serial number (**serial#**) from v$session for the SH user, and then describe the DBMS_MONITOR package. Then, from the SYSDBA session, enable tracing using the sid and serial# values for the other session, including the waits and bind information, with the lab_08_10.sql script:

12. From the SH session, execute the **lab_08_11.sql** script, and then exit your session.

13. From the remaining SYSDBA session, determine your USER_DUMP_DEST location, locate the trace file, and view the contents. Determine the location of the trace files by using the SHOW PARAMETER DUMP command and making note of the UDUMP destination.

14. Exit your session.

15. Change directory to the UDUMP destination that you retrieved by the previous query.

16. Locate your file.

17. View the top file that is the most recent.

18. Convert the file to readable format using the TKPROF utility.

19. You can use gedit to view the file. Change back to your home directory and then to the **labs** directory.

# Appendix B: Workshops

**Frequently Used Scripts: Reference**

| Script name | Description |
|---|---|
| `rp.sql` | Script to retrieve information from the plan table using `dbms_xplan` |
| `li.sql` | Lists all indexes; accepts table name as argument; wildcards (%,_) in table names are allowed (Default: Previous table name) |
| `ci.sql` | Creates an index; prompts for table name and column names<br>(Column names should be separated with commas.) |
| `cbi.sql` | Creates a bitmap index on partitioned table; prompts for table name and column names (Column names should be separated with commas.) |
| `cbinp.sql` | Creates a bitmap index on nonpartitioned table; prompts for table name and column names (Column names should be separated with commas.) |
| `cui.sql` | Creates a unique index; same behavior as `ci.sql` |
| `aton.sql` | `Set autotrace on` |
| `atonx.sql` | `Set autotrace on explain` |
| `atto.sql` | `Set autotrace traceonly` |
| `attox.sql` | `Set autotrace traceonly explain` |
| `atoff.sql` | `Set autotrace off` |
| `di.sql` | `Drops a named index` |
| `dai.sql` | `Drops all indexes for a given table` |

## Workshop 1

## Part A: Writing Good SQL

**Objectives:**

- Learn about inequality conditions
- Learn about equality conditions
- Learn about NULL usage
- Learn about the effects of functions on index usage
- Learn to tune sorts for ORDER BY clauses
- Identify and tune implicit sort operations caused by SELECT DISTINCT
- Learn to tune GROUP BY operations and group functions
- Learn to tune set operators (UNION, MINUS, INTERSECT)

This workshop can be done entirely in the SQL*Plus environment. Try to work in small groups and discuss the workshop results. Each time you load a new SQL statement, try to predict what the optimizer will do before running the statement. Take notes during the workshop as an aid for the wrap-up discussion.

1. Open a terminal window. Change directory to workshop directory. Log in to SQL*Plus as SH with password SH. Use the **attox.sql** script to enable AUTOTRACE TRACEONLY EXPLAIN to suppress statement output and produce execution plans; check the settings with SHOW AUTOTRACE.

```
$cd workshop
$ sqlplus sh/sh
SQL> show autotrace
SQL> @attox
SQL> show autotrace
Student Observations:
```

2. Get and run queries **ws_01_01.sql**, **ws_01_01a.sql**, **ws_01_02.sql**, **ws_01_02b.sql**, and **ws_01_03.sql**. First, remove all indexes from the CUSTOMERS table by running the **dai.sql** script.

**Note:** dai.sql removes all nonprimary key indexes.

**Note:** The CUST_ID column is the primary key and will still be indexed.

```
SQL> @dai
on which table: customers

SQL> get ws_01_01
  1  SELECT cust_first_name
  2 , cust_last_name
  3 FROM customers
  4* WHERE cust_id = 1030
SQL>@ws_01_01
SQL> get ws_01_01a
```

```
  1 SELECT cust_first_name
  2 , cust_last_name
  3 FROM customers
  4* WHERE cust_id <> 1030
SQL>@ws_01_01a
SQL> get ws_01_02
  1 SELECT cust_first_name
  2 , Cust_Last_Name
  3 FROM customers
  4* WHERE cust_id < 10
SQL>@ws_01_02
SQL> get ws_01_02b
  1 SELECT cust_first_name
  2 , cust_last_name
  3 FROM customers
  4* WHERE cust_id < 10000
SQL>@ws_01_02b
SQL> get ws_01_03
  1 SELECT cust_first_name
  2 , cust_last_name
  3 FROM customers
  4* WHERE cust_id between 70 AND 80
SQL>@ws_01_03
```

Analyze the results of these queries and determine when the Oracle optimizer can use indexes.

Indexes may be used for three types of conditions:

- Equality search (ws_01_01.sql)
- Unbounded range (ws_01_02 and ws_01_02b.sql)
- Bounded range (ws_01_03.sql)

However, even then the optimizer considers the selectivity of the operation before using an index. (If you have time, try changing the values in **ws_01_03.sql** to 70000 and 80000 and see what happens.) The index is not used if the NOT EQUAL (<>) operator is present.

**Note:** Index usage may be forced using an INDEX hint.

**Student Observations:**

3. Now create an index on the CUST_CREDIT_LIMIT column of the CUSTOMERS table by using the **ci.sql** script. Explain the queries in **ws_01_04.sql** and **ws_01_05.sql**. Use the **rp.sql** script to retrieve the information from PLAN_TABLE by using the dbms_xplan package. Make sure that you have disabled AUTOTRACE by running **atoff.sql**.

```
SQL> @atoff
SQL> @ci
on which table : customers
on which column(s): cust_credit_limit
Creating index on: customers cust_credit_limit
SQL> get ws_01_04
  1 explain plan for
  2  SELECT cust_id
  3 FROM customers
  4 WHERE cust_credit_limit*1.10 = 11000
SQL>@ws_01_04
SQL> @rp
SQL> get ws_01_05
  1 explain plan for
  2 SELECT cust_id
  3 FROM customers
  4 WHERE cust_credit_limit = 30000/2
SQL>@ws_01_05
SQL> @rp
```

The results show that although the CUST_CREDIT_LIMIT column is indexed, the index is not used by default. This can happen if the indexed column is part of an expression in the WHERE clause. An index is usable only if the indexed column appears clean in the WHERE clause and even then may only be used based on selectivity. The CUST_CREDIT_LIMIT column has poor selectivity because it has only eight values, which may result in the index being ignored at times.

Notice how the optimizer filters on 15000 automatically.

**Student Observations:**

4. Create an index on the CUST_LAST_NAME column of the CUSTOMERS table. View the explain plan for queries **ws_01_06.sql**, **ws_01_07.sql**, **ws_01_07a.sql**, **ws_01_07b.sql**, and **ws_01_07c.sql** by using **rp.sql**.

```
SQL> @ci
on which table : customers
on which column(s): cust_last_name
SQL> get ws_01_06
  1 explain plan for
  2 SELECT cust_id
  3 FROM customers
  4 WHERE SUBSTR(cust_last_name,1,1) = 'S'
SQL>@ws_01_06
SQL> @rp
SQL> get ws_01_07
  1 explain plan for
  2 SELECT cust_id
  3 FROM customers
  4 WHERE cust_last_name like 'S%'
SQL>@ws_01_07
SQL> @rp
SQL> get ws_01_07a
  1 explain plan for
  2 SELECT cust_id
  3 FROM customers
  4 WHERE upper(cust_last_name) ='KING'
SQL>@ws_01_07a
SQL> @rp
SQL> get ws_01_07b
  1 explain plan for
  2 SELECT cust_id
  3 FROM customers
  4 WHERE cust_last_name ='King'
SQL>@ws_01_07b
SQL> @rp
SQL> get ws_01_07c
  1 explain plan for
  2 SELECT cust_id
  3 FROM customers
  4 WHERE cust_last_name = initcap('KING')
SQL>@ws_01_07c
SQL> @rp
```

You see that how you write your SQL has a significant impact on whether indexes are used or not. The indexed column should be clean. As soon as you apply any function, index usage does not happen by default. Also an index may not be used for selectivity reasons. You may, however, force index usage with hints.

**Student Observations:**

5. Now start query **`ws_01_08.sql`** and compare the execution plan with **`ws_01_07.sql`**. How do you explain the difference?

```
SQL> get ws_01_08
  1  explain plan for
  2  SELECT cust_last_name
  3  FROM customers
  4  WHERE cust_last_name like 'S%'
SQL>@ws_01_08
SQL> @rp
```

**Student Observations:**

6. Create the explain plan for the query in **`ws_01_09.sql`**. Then use the **`rp.sql`** script to retrieve the information in a meaningful way from `PLAN_TABLE`. Try to explain why the index is not used: The indexed column is clean, and the search pattern does not start with a wildcard.
   **Hint:** Look at the filter.

```
SQL> get ws_01_09
  1 explain plan for
  2 SELECT cust_last_name
  3 FROM customers
  4 WHERE cust_id like '7%'
```

```
SQL>@ws_01_09
SQL>@rp
```

**Student Observations:**

7. In this exercise, you investigate the treatment of NULL-values and use the CUSTOMERS table for that purpose. First, run the **ws_01_10a.sql** script to remove some values from the CUST_EMAIL column. Second, create an index on the CUST_EMAIL column of the CUSTOMERS table; this column contains many NULL-values. Then start query **ws_01_10b.sql**.

```
SQL> @atoff
SQL> @ws_01_10a
SQL> describe customers
SQL> @ci
on which table : customers
on which column(s): cust_email
SQL> get ws_01_10b
  1  explain plan for
  2  SELECT cust_email
  3 FROM customers
  4* WHERE cust_email is null
SQL>@ws_01_10b
SQL>@rp
```

**Student Observations:**

To explain why the index is not used in this case, you should be aware that the Oracle Server does not store any references to NULL-values in a regular B*-tree index. That is why the only way to find rows containing NULL values is to perform a full table scan. In the case of a concatenated index, if all columns in the index are NULL, then the same explanation applies.

**Student Observations:**

8. Now suppose that you are interested in all rows that do *not* contain a NULL value. Start **ws_01_11.sql** and compare with **ws_01_10b.sql**.

```
SQL> get ws_01_11
  1  explain plan for
SELECT cust_id
FROM customers
  4* WHERE cust_email is not null
SQL>@ws_01_11
SQL> @rp
```

The index is again not used but for a different reason: The optimizer decision is based on selectivity considerations. You can assist the optimizer in choosing a fast full index scan by creating a concatenated index on the `CUST_ID` and `CUST_EMAIL` columns.

```
SQL> @ci
on which table : customers
on which column(s): cust_id, cust_email
SQL> @ws_01_11
SQL> @rp
```

**Student Observations:**

## Part B: Sorting, Grouping, and Set Operators

In this workshop, you concentrate on tuning explicit and implicit sort operations. Sorting is a common operation. If the Oracle Server is able to perform all sort activity in memory, then the performance is probably acceptable. However, sometimes the Oracle Server writes intermediate results to disk. Therefore, SQL*Plus AUTOTRACE shows two statistics: sorts (memory) and sorts (disk). Sometimes you can avoid sort operations by creating indexes, or you can suppress implicit sorts that are not needed for the result that you want.

1. Drop all indexes on the CUSTOMERS table using the **dai.sql** script. Note that the index that is related to the primary key constraint cannot (and does not need to) be dropped. View and execute query **ws_01_13.sql**:

```
SQL> @dai on which table: customers
SQL> get ws_01_13.sql
  1  explain plan for SELECT cust_first_name
  2  , cust_last_name
  3  , cust_credit_limit
  4  FROM customers
  5* ORDER BY cust_credit_limit
SQL>@ws_01_13
SQL>@rp
```

**Student Observations:**

The Oracle Server must perform a sort operation.

2. Sorting can be avoided by creating appropriate indexes, so investigate what happens when you create an index on the CUST_CREDIT_LIMIT column:

```
SQL> @ci
on which table : customers
on which column(s): cust_credit_limit
Creating index on: customers cust_credit_limit
SQL> @ws_01_13
SQL> @rp
```

The results show that although the CUST_CREDIT_LIMIT column is indexed, the optimizer does not use the index (to avoid a sort operation).

**Student Observations:**

3. Get and run **ws_01_14.sql** and compare the results with **ws_01_13.sql**.

```
SQL> get ws_01_14
  1  explain plan for SELECT cust_first_name
  2 , cust_last_name
  3 , cust_credit_limit
  4 FROM customers
  5* ORDER BY cust_id
SQL>@ws_01_14
SQL> @rp
```

Is the index on CUST_ID used?

**Student Observations:**

4. Get and run ws_**01_15.sql**. Does the WHERE clause make a difference?

```
SQL> get ws_01_15
  1  explain plan for SELECT cust_first_name
  2  , cust_last_name
  3  , cust_city
  4  FROM customers
  5  WHERE cust_city = 'Paris'
  6* ORDER BY cust_id
SQL>@ws_01_15
SQL> @rp
```

**Student Observations:**

5. Investigate what happens if you create an additional index on the CUST_CITY column:

```
SQL> @ci
on which table : customers
on which column(s): cust_city

SQL> @li
List indexes on table: customers
SQL> @ ws_01_15
SQL> @rp
```

Which index is used?

**Student Observations:**

6. Now get and run queries **ws_01_16.sql**, **ws_01_17.sql**, and **ws_01_18.sql**.

```
SQL> get ws_01_16
  1  explain plan for SELECT max(cust_credit_limit)
  2* FROM customers
SQL>@ws_01_16
SQL> @rp
SQL> get ws_01_17
  1  explain plan for SELECT max(cust_credit_limit+1000)
  2* FROM customers
SQL>@ws_01_17
SQL> @rp
SQL> get ws_01_18
  1  explain plan for SELECT max(cust_credit_limit*2)
  2* FROM customers
SQL>@ws_01_18
SQL> @rp
```

This shows that an index can be useful to retrieve a maximum value (and a minimum value). If no index is available, the optimizer must scan the full table and perform a sort. Sometimes an operation on the indexed column value (such as in **`ws_01_18.sql`**) prevents the index from being used. If you have time, try creating an index on `(CUST_CREDIT_LIMIT*2)`. Does this help?

**Student Observations:**

7. Start query **`ws_01_19.sql`**. A `WHERE` clause is added and you see the result.

```
SQL> get ws_01_19
  1  explain plan for SELECT max(cust_credit_limit)
  2  FROM customers
  3* WHERE cust_city = 'Paris'
SQL>@ws_01_19
SQL> @rp
```

**Student Observations:**

8. Start query **`ws_01_20.sql`**. This query shows that to evaluate a `SELECT DISTINCT`, the index is used.

```
SQL> get ws_01_20
  1  explain plan for SELECT distinct cust_city
  2* FROM customers
SQL>@ws_01_20
SQL> @rp
```

**Student Observations:**

Drop the index on the `CUST_CITY` column and run **`ws_01_19.sql`** again.

```
SQL> @li
List indexes on table: customers
SQL> drop index cust_cust_city_idx;
SQL> @ws_01_19
SQL> @rp
```

**Student Observations:**

The optimizer chooses a full table scan in the absence of the index. Now run **`ws_01_20.sql`** and obtain the explain plan.

```
SQL> @ws_01_20
SQL> @rp
```

**Student Observations:**

9. The following queries investigate the SQL set operators. These operators unconditionally result in sort operations, regardless of the presence of indexes. To investigate this, create any indexes you like. The sorts are needed because the SQL set operators are supposed to filter duplicate rows from the result.

```
SQL> get ws_01_21
  1  explain plan for
  2  SELECT cust_last_name FROM customers
  3  Where cust_city = 'Paris'
  4  intersect
  5  SELECT cust_last_name FROM customers
  6* Where cust_credit_limit < 10000
SQL>@ws_01_21
SQL> @rp

SQL> get ws_01_22
  1  explain plan for
  2  SELECT cust_last_name FROM customers
  3  Where cust_city = 'Paris'
  4  minus
  5  SELECT cust_last_name FROM customers
  6* Where cust_credit_limit > 10000
SQL>@ws_01_22
SQL> @rp
SQL> get ws_01_23
  1  explain plan for
  2  SELECT cust_last_name FROM customers
  3  Where cust_city = 'Paris'
  4  UNION
  5  SELECT cust_last_name FROM customers
  6* Where cust_credit_limit < 10000
SQL>@ws_01_23
SQL> @rp
```

**Student Observations:**

There is one exception: The UNION ALL operator does not perform a sort and does not filter duplicate rows. Use the UNION ALL operator if you are sure that there are no duplicate rows or that duplicate rows cause no semantic problems.

```
SQL> get ws_01_24
  1  explain plan for
  2  SELECT cust_last_name FROM customers
  3  Where cust_city = 'Paris'
  4  UNION ALL
  5  SELECT cust_last_name FROM customers
  6* Where cust_credit_limit < 10000
SQL>@ws_01_24
SQL> @rp
```

**Student Observations:**

10. Examine the GROUP BY operator using the **ws_01_25.sql** script. Try indexing the GROUP BY column. What happens?

```
SQL> get ws_01_25
  1  explain plan for SELECT cust_city
  2 , avg(cust_credit_limit)
  3 FROM customers
  4* GROUP BY cust_city
SQL>@ws_01_25
SQL> @rp
SQL> @ci
on which table : customers
on which column(s): cust_city
Creating index cust_cust_city_idx
SQL> @ws_01_25
SQL> @rp
```

**Student Observations:**

Now examine the following two SQL statements, which are logically equivalent:

```
SQL> get ws_01_26
  1  explain plan for SELECT cust_city
  2  , avg(cust_credit_limit)
  3  FROM customers
  4  WHERE cust_city = 'Paris'
  5* GROUP BY cust_city
SQL>@ws_01_26
SQL> @rp
Student Observations:
SQL> get ws_01_27
  1 explain plan for
  2 SELECT cust_city
  3  , avg(cust_credit_limit)
  4  FROM customers
  5  GROUP BY cust_city
  6* having cust_city = 'Paris'
SQL>@ws_01_27
SQL> @rp
```

The index on the CUST_CITY column may be used in **ws_01_26.sql** to reduce the set of rows that must be sorted. This is not possible in **ws_01_27.sql**, because the HAVING clause is always evaluated after the GROUP BY clause. Note that **ws_01_27.sql** is a badly formulated SQL statement. A HAVING clause usually contains a group function: COUNT, SUM, AVG, MIN, or MAX.

**Student Observations:**

### Part C: Subqueries

11. Log in as **HR/HR**. Set AUTOTRACE to TRACE ONLY.

```
SQL> @atto
```

12. View and execute the SQL statement in **`ws_01_28.sql`**.

```
SQL> get ws_01_28
  1  SELECT employee_id, last_name
  2  FROM   employees e1
  3  WHERE  e1.salary >
  4         (SELECT avg(salary)
  5          FROM employees e2
  6          WHERE e1.department_id = e2.department_id
  7*         GROUP BY e2.department_id)
SQL>@ws_01_28
```

This correlated subquery example processes all the rows of the outer table (C1). Each row in the outer table is checked against every row from the inner condition.

**Student Observations:**

If you reformulate the query to use a join between two tables, one of which is built up on the spot (an in-line view), does performance improve? Use the **`ws_01_29.sql`** script for this purpose.

```
SQL> get ws_01_29
  1  SELECT e1.employee_id, e1.last_name
  2  FROM   employees e1
  3  ,      (SELECT avg(salary) avg_sal, department_id
  4          FROM   employees
  5          GROUP BY department_id) a
  6  WHERE  e1.salary > a.avg_sal
  7* AND    e1.department_id = a.department_id
SQL> @ws_01_29
```

**Student Observations:**

## Workshop Summary

After completing this workshop, you should have learned the following:

- Indexes can be used to avoid sorting. However, the optimizer must not miss any rows, which is why the index must be on a `NOT NULL` column.
- If any additional indexes are available to reduce the set of rows to be sorted, those paths may be more effective. Sorts on small sets of rows usually perform well, and this approach reduces throwaway (retrieving rows that are not needed for the result set).
- Maximum and minimum values can be retrieved from indexes, with certain restrictions (for example, as long as there is no `WHERE` clause and no `GROUP BY` clause).
- `SELECT DISTINCT`, `GROUP BY`, `UNION`, `MINUS`, and `INTERSECT` all result in an unconditional sort operation. This sort operation cannot be suppressed. However, it can be tuned by trying to reduce the set of rows to be sorted.
- Try to use the `WHERE` clause for row-level predicates instead of the `HAVING` clause.

## Workshop 2

## Part A : Identifying Problematic SQL

### Workshop Objectives

- Using ADDM to identify problematic SQL
- Using Top SQL to identify high load SQL

### Using ADDM

1. Open a terminal window. Change directory to workshop directory. Log in to SQL*Plus as **HR/HR** and execute **`ws_02_01.sql`**.

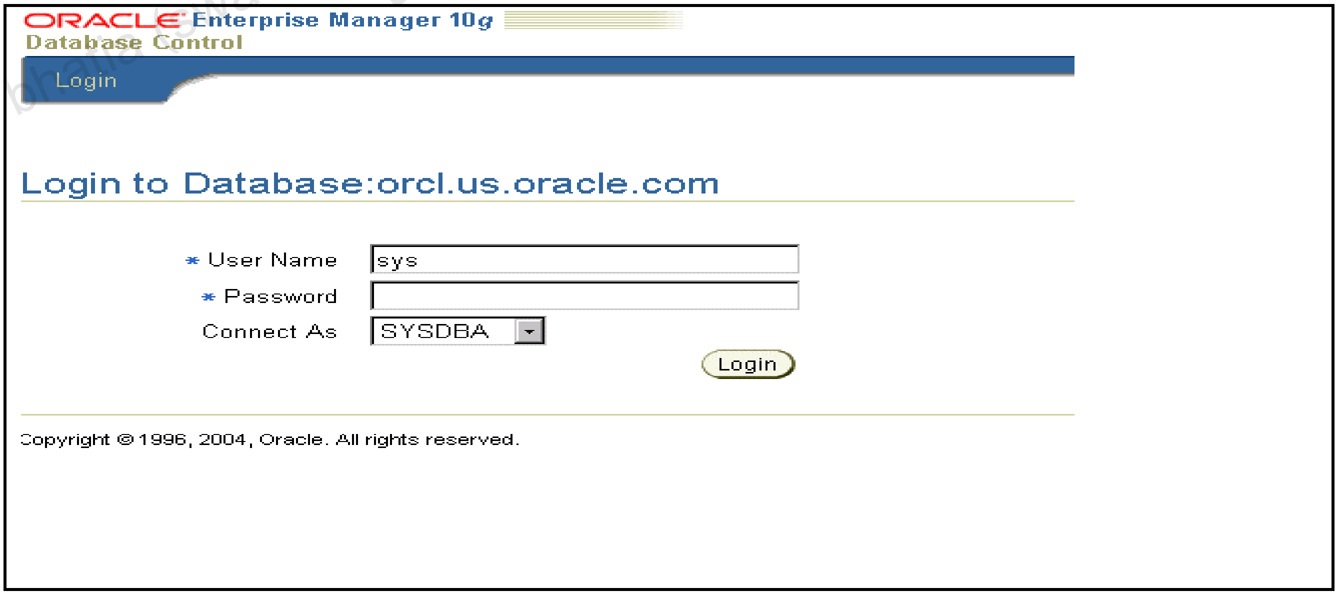
```
SQL>get ws_02_01
1 create table emp as SELECT * FROM employees;
2 delete emp;
SQL>@ws_02_01
```

2. Open another terminal window and execute the following commands to create a row locking conflict:

```
SQL>sqlplus hr/hr
SQL>delete emp;
```

3. Open your browser and enter the following URL:

```

http://<hostname>:1158/em
Enter sys/<password> as SYSDBA and click Login.
```



Login to Database:orcl.us.oracle.com

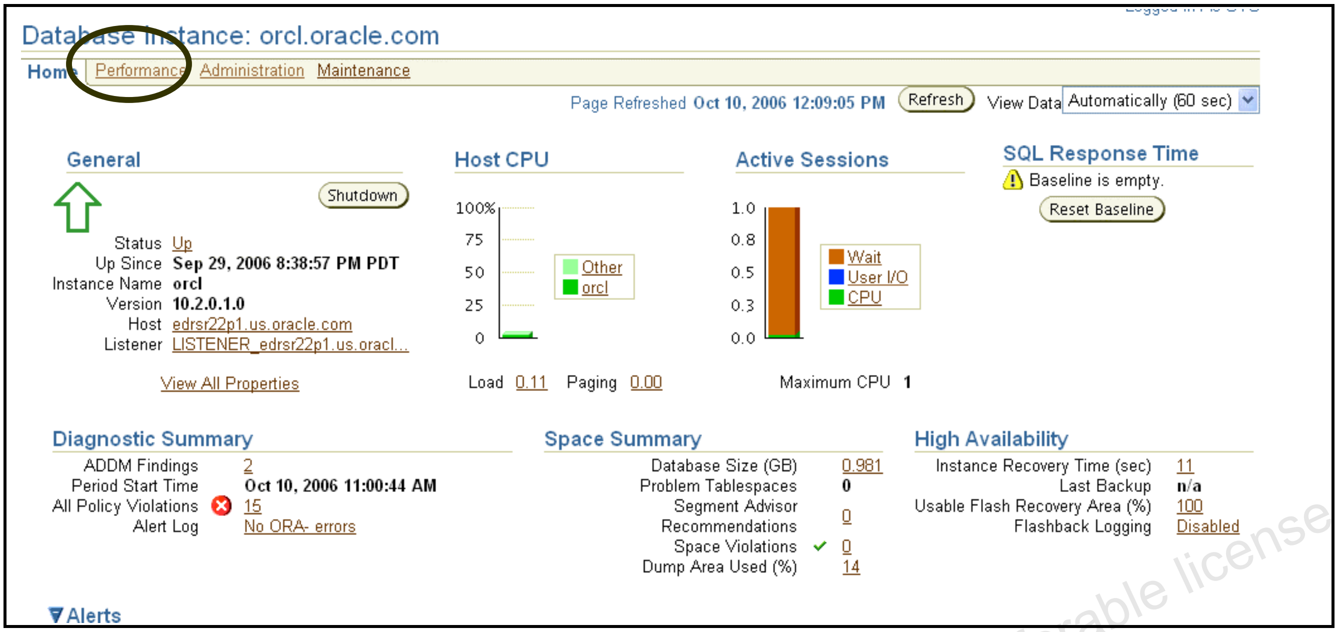
* User Name: sys

* Password:

Connect As: SYSDBA

Copyright © 1996, 2004, Oracle. All rights reserved.

4. From your Enterprise Manager browser window, click the **Performance** tab.



Database Instance: orcl.oracle.com

Home | Performance | Administration | Maintenance

Page Refreshed Oct 10, 2006 12:09:05 PM (Refresh) View Data Automatically (60 sec)

**General**

Status Up
Up Since Sep 29, 2006 8:38:57 PM PDT
Instance Name orcl
Version 10.2.0.1.0
Host edrsr22p1.us.oracle.com
Listener LISTENER_edrsr22p1.us.oracl...

View All Properties

**Host CPU**

Load 0.11 Paging 0.00

**Active Sessions**

Maximum CPU 1

**SQL Response Time**

Baseline is empty.

**Diagnostic Summary**

ADDM Findings 2
Period Start Time Oct 10, 2006 11:00:44 AM
All Policy Violations 15
Alert Log No ORA- errors

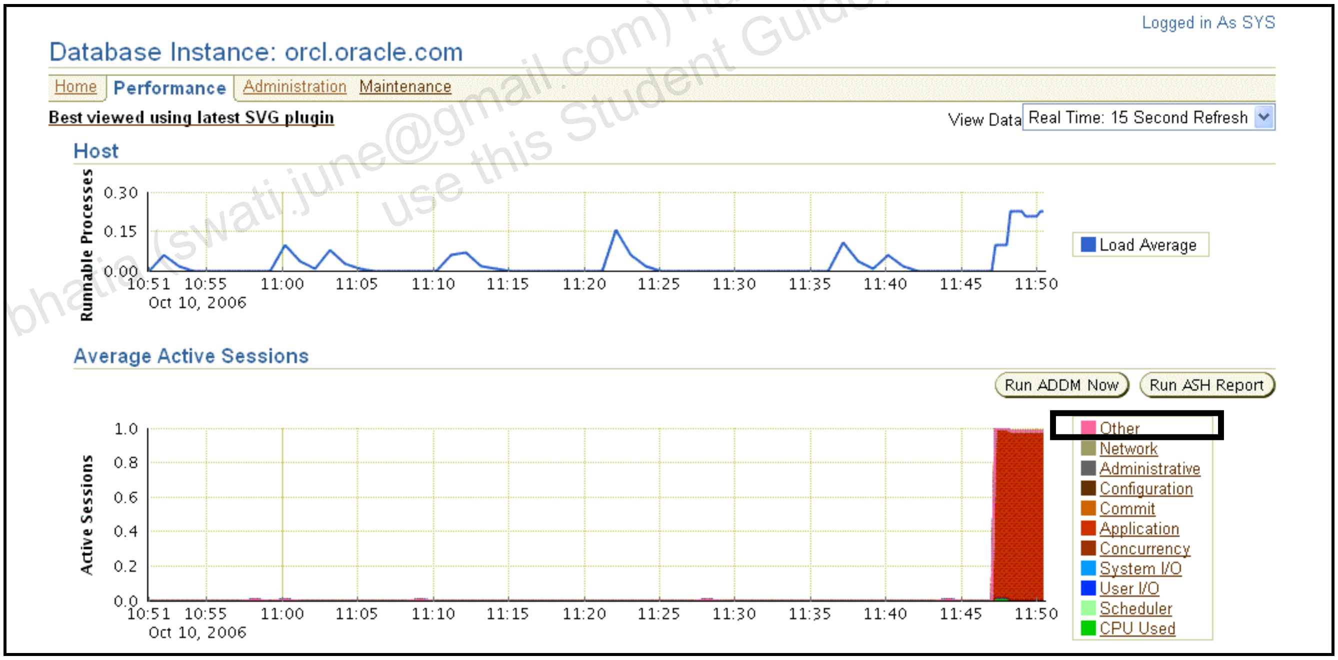
**Space Summary**

Database Size (GB) 0.981
Problem Tablespaces 0
Segment Advisor Recommendations 0
Space Violations 0
Dump Area Used (%) 14

**High Availability**

Instance Recovery Time (sec) 11
Last Backup n/a
Usable Flash Recovery Area (%) 100
Flashback Logging Disabled

5. Give the screen a chance to refresh for a couple of minutes. You see that the Active sessions graph shows high activity. Click Other to see what is causing it. You can see that sessions are waiting. Go back to the Performance page and after about **10 minutes** scroll to the **bottom** of the window.

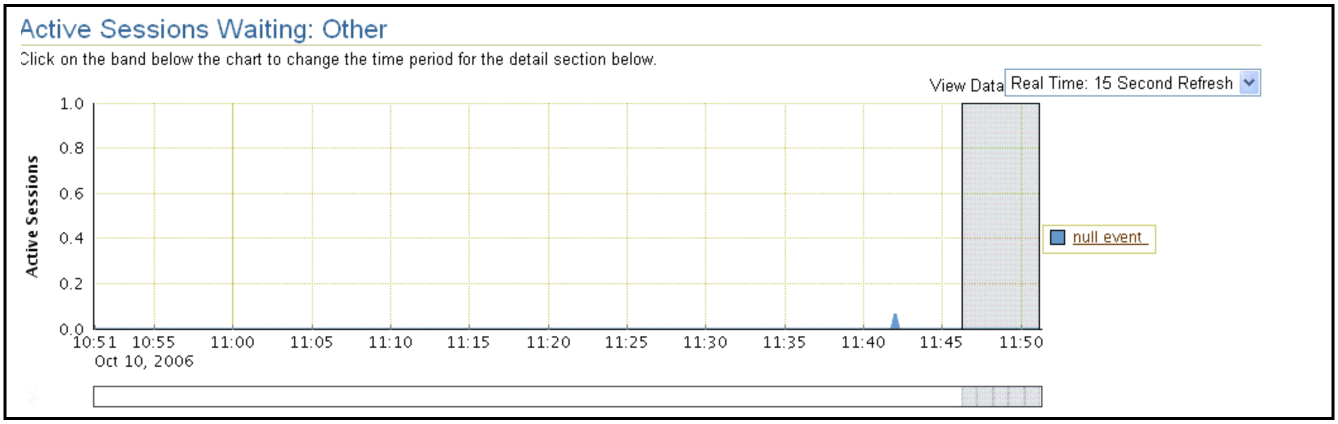


Database Instance: orcl.oracle.com

Home | Performance | Administration | Maintenance

Best viewed using latest SVG plugin

View Data Real Time: 15 Second Refresh

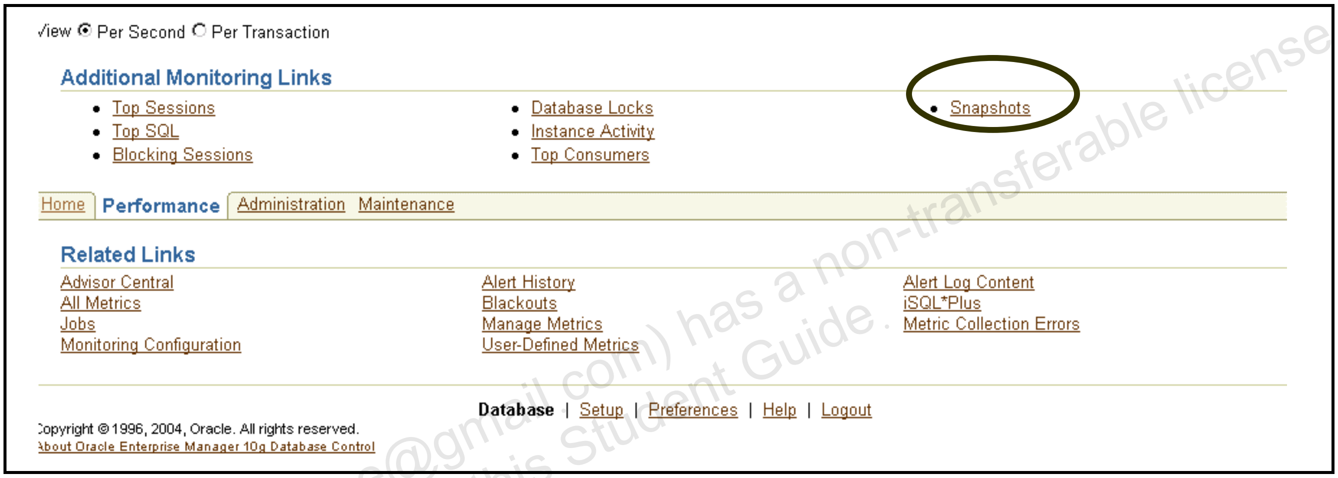


Active Sessions Waiting: Other

Click on the band below the chart to change the time period for the detail section below.

View Data Real Time: 15 Second Refresh

6. You want to create a snapshot to capture the performance finding. Click the **Snapshots** link.



View ● Per Second ○ Per Transaction

Additional Monitoring Links

- Top Sessions
- Top SQL
- Blocking Sessions
- Database Locks
- Instance Activity
- Top Consumers
- Snapshots

Home | Performance | Administration | Maintenance

Related Links

Advisor Central<br>All Metrics<br>Jobs<br>Monitoring Configuration

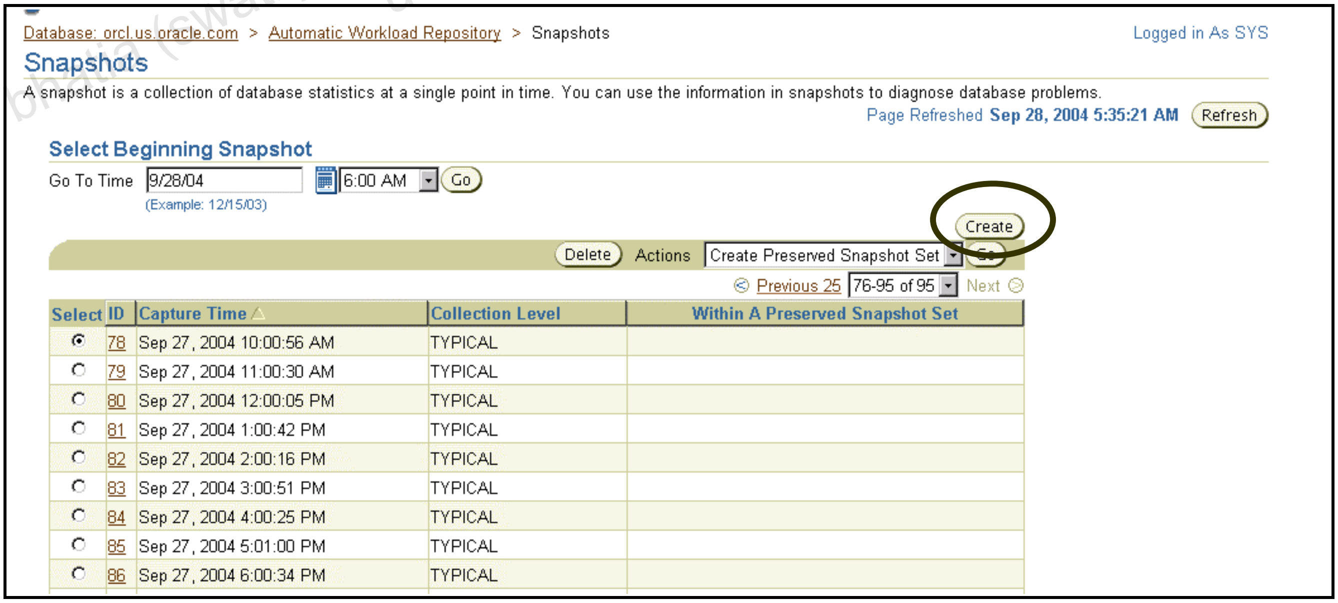
Alert History<br>Blackouts<br>Manage Metrics<br>User-Defined Metrics

Alert Log Content<br>iSQL*Plus<br>Metric Collection Errors

Database | Setup | Preferences | Help | Logout

Copyright © 1996, 2004, Oracle. All rights reserved.
About Oracle Enterprise Manager 10g Database Control

7. Click the **Create** button to create a snapshot.



Database: orcl.us.oracle.com > Automatic Workload Repository > Snapshots

Logged in As SYS

Snapshots

A snapshot is a collection of database statistics at a single point in time. You can use the information in snapshots to diagnose database problems.

Page Refreshed **Sep 28, 2004 5:35:21 AM** Refresh

Select Beginning Snapshot

Go To Time 9/28/04 6:00 AM Go
(Example: 12/15/03)

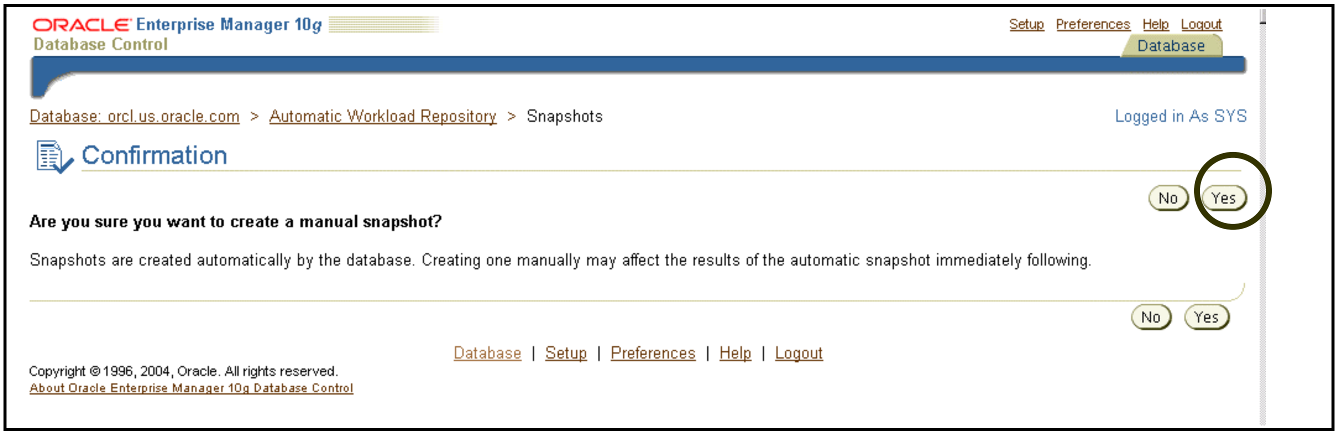
Create

Delete Actions Create Preserved Snapshot Set Go

Previous 25 76-95 of 95 Next

| Select | ID | Capture Time | Collection Level | Within A Preserved Snapshot Set |
|---|---|---|---|---|
| ● | 78 | Sep 27, 2004 10:00:56 AM | TYPICAL | |
| ○ | 79 | Sep 27, 2004 11:00:30 AM | TYPICAL | |
| ○ | 80 | Sep 27, 2004 12:00:05 PM | TYPICAL | |
| ○ | 81 | Sep 27, 2004 1:00:42 PM | TYPICAL | |
| ○ | 82 | Sep 27, 2004 2:00:16 PM | TYPICAL | |
| ○ | 83 | Sep 27, 2004 3:00:51 PM | TYPICAL | |
| ○ | 84 | Sep 27, 2004 4:00:25 PM | TYPICAL | |
| ○ | 85 | Sep 27, 2004 5:01:00 PM | TYPICAL | |
| ○ | 86 | Sep 27, 2004 6:00:34 PM | TYPICAL | |

8. Click the **Yes** button to create a manual snapshot.



ORACLE Enterprise Manager 10*g*
Database Control

Setup Preferences Help Logout
Database

Database: orcl.us.oracle.com > Automatic Workload Repository > Snapshots

Logged in As SYS

Confirmation

No [✓ Yes]

**Are you sure you want to create a manual snapshot?**

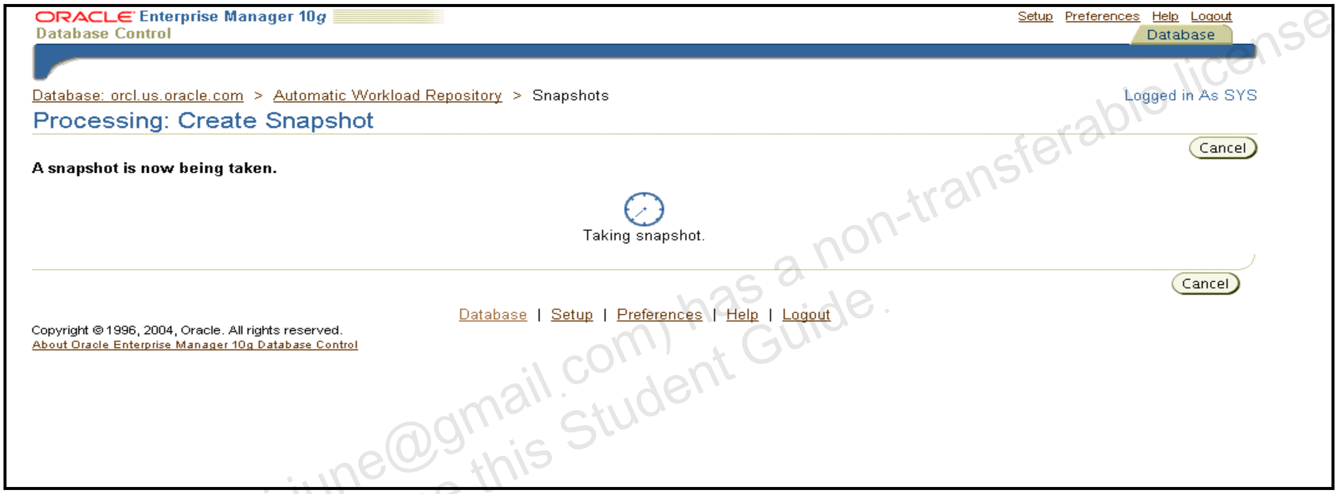
Snapshots are created automatically by the database. Creating one manually may affect the results of the automatic snapshot immediately following.

No Yes

Database | Setup | Preferences | Help | Logout

Copyright © 1996, 2004, Oracle. All rights reserved.
About Oracle Enterprise Manager 10g Database Control

9. A snapshot is now being taken.



ORACLE Enterprise Manager 10*g*
Database Control

Setup Preferences Help Logout
Database

Database: orcl.us.oracle.com > Automatic Workload Repository > Snapshots

Logged in As SYS

Processing: Create Snapshot

Cancel

**A snapshot is now being taken.**

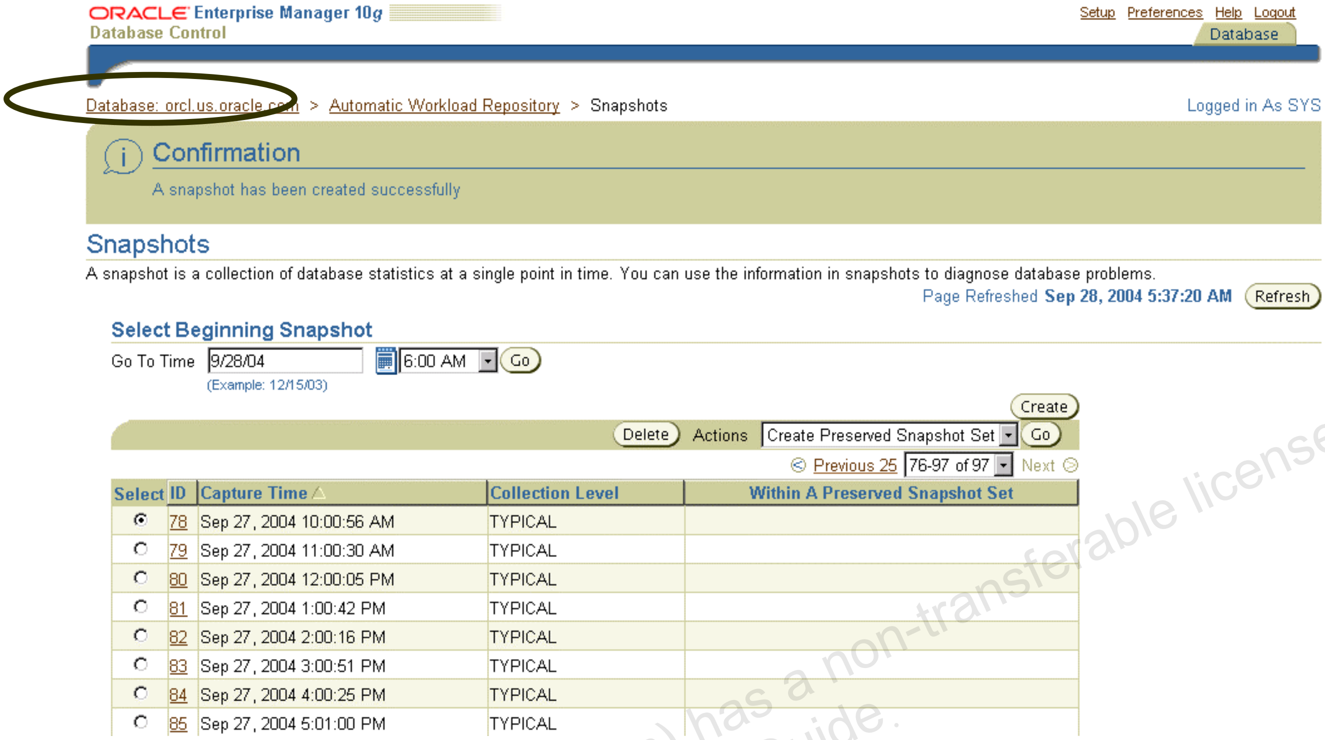
Taking snapshot.

Cancel

Database | Setup | Preferences | Help | Logout

Copyright © 1996, 2004, Oracle. All rights reserved.
About Oracle Enterprise Manager 10g Database Control

10. When the snapshot is created, click the **Database** breadcrumb and then the **Home** tab.



Database: orcl.us.oracle.com > Automatic Workload Repository > Snapshots

Logged in As SYS

**Confirmation**

A snapshot has been created successfully

**Snapshots**

A snapshot is a collection of database statistics at a single point in time. You can use the information in snapshots to diagnose database problems.

Page Refreshed Sep 28, 2004 5:37:20 AM  Refresh

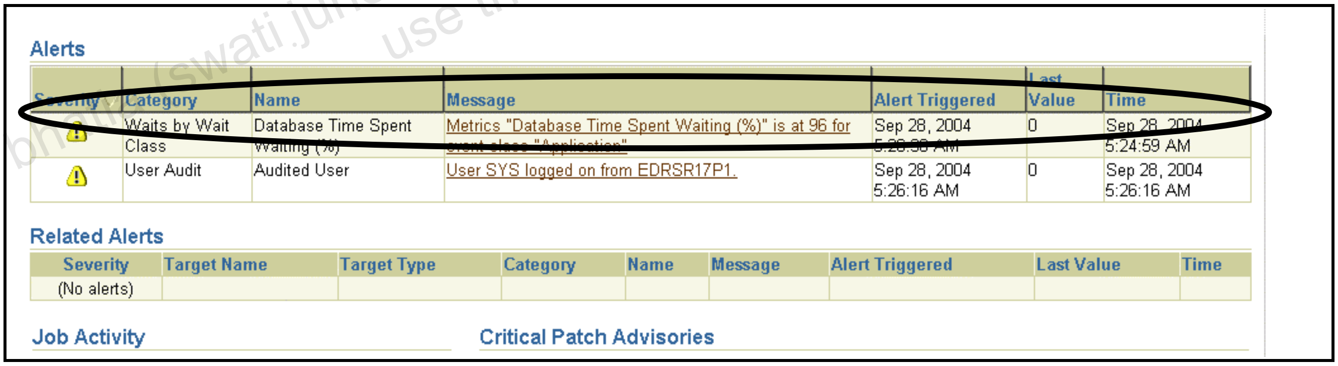
**Select Beginning Snapshot**

Go To Time 9/28/04  6:00 AM  Go
(Example: 12/15/03)

Create
Delete  Actions  Create Preserved Snapshot Set  Go
Previous 25  76-97 of 97  Next

| Select | ID | Capture Time | Collection Level | Within A Preserved Snapshot Set |
|---|---|---|---|---|
| ● | 78 | Sep 27, 2004 10:00:56 AM | TYPICAL | |
| ○ | 79 | Sep 27, 2004 11:00:30 AM | TYPICAL | |
| ○ | 80 | Sep 27, 2004 12:00:05 PM | TYPICAL | |
| ○ | 81 | Sep 27, 2004 1:00:42 PM | TYPICAL | |
| ○ | 82 | Sep 27, 2004 2:00:16 PM | TYPICAL | |
| ○ | 83 | Sep 27, 2004 3:00:51 PM | TYPICAL | |
| ○ | 84 | Sep 27, 2004 4:00:25 PM | TYPICAL | |
| ○ | 85 | Sep 27, 2004 5:01:00 PM | TYPICAL | |

11. A performance finding is now detected through an alert in the **Alerts** section of the **Home** page.



**Alerts**

| Severity | Category | Name | Message | Alert Triggered | Last Value | Time |
|---|---|---|---|---|---|---|
| ⚠ | Waits by Wait Class | Database Time Spent Waiting (%) | Metrics "Database Time Spent Waiting (%)" is at 96 for event class "Application" | Sep 28, 2004<br>[illegible] AM | 0 | Sep 28, 2004<br>5:24:59 AM |
| ⚠ | User Audit | Audited User | User SYS logged on from EDRSR17P1. | Sep 28, 2004<br>5:26:16 AM | 0 | Sep 28, 2004<br>5:26:16 AM |

**Related Alerts**

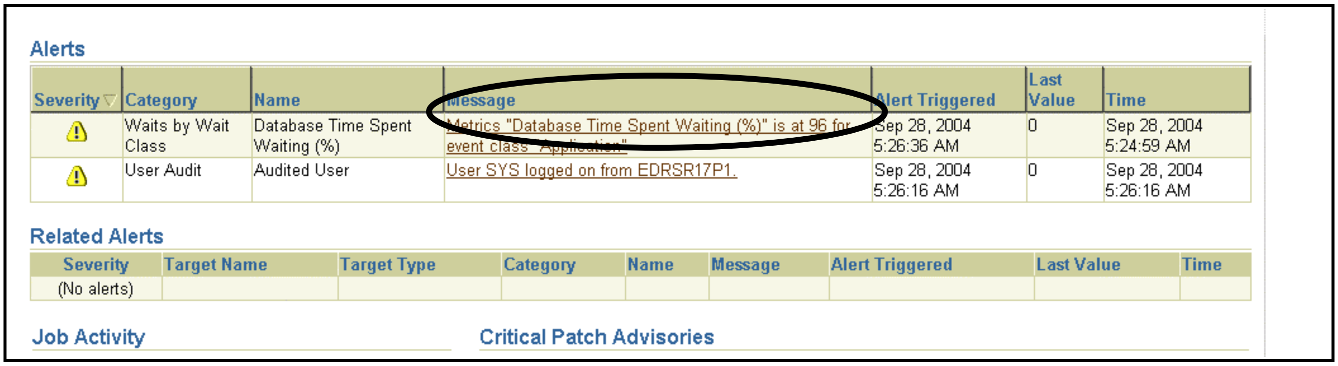
| Severity | Target Name | Target Type | Category | Name | Message | Alert Triggered | Last Value | Time |
|---|---|---|---|---|---|---|---|---|
| (No alerts) | | | | | | | | |

**Job Activity**  **Critical Patch Advisories**

## Resolving the Performance Finding by Using ADDM

When a performance finding is encountered, you can use ADDM to resolve it. Perform the following steps:

1. Click the Alert **Database Time Spent Waiting**.



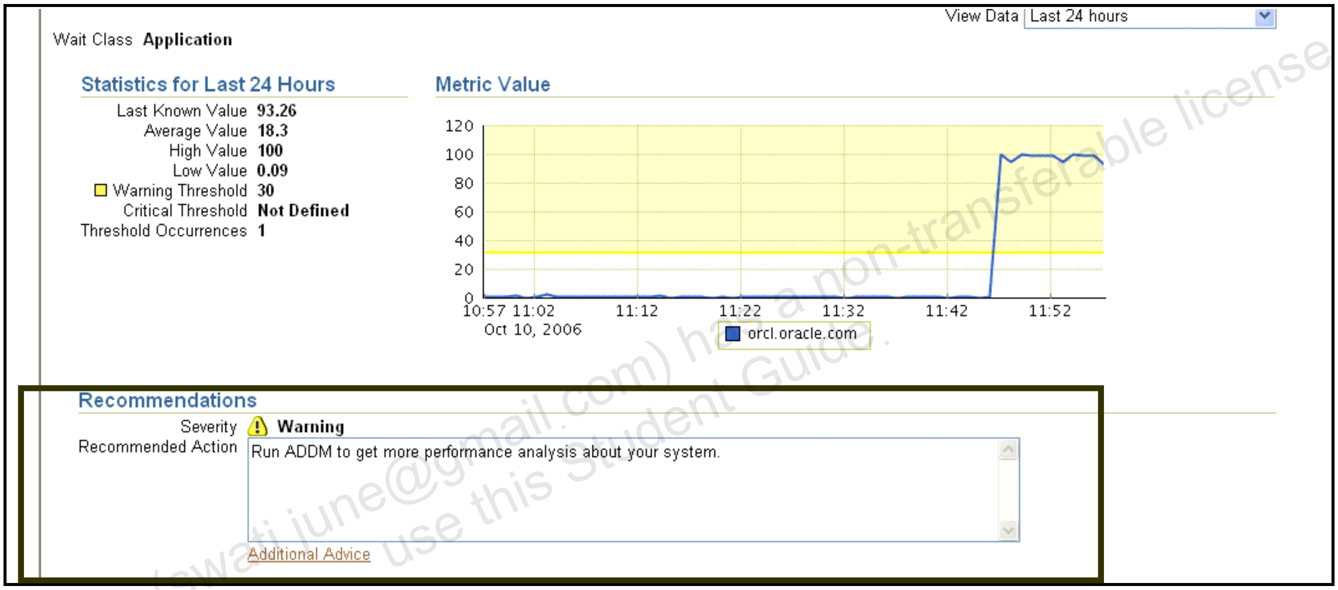
**Alerts**

| Severity | Category | Name | Message | Alert Triggered | Last Value | Time |
|---|---|---|---|---|---|---|
| ⚠ | Waits by Wait Class | Database Time Spent Waiting (%) | Metrics "Database Time Spent Waiting (%)" is at 96 for event class "Application" | Sep 28, 2004 5:26:36 AM | 0 | Sep 28, 2004 5:24:59 AM |
| ⚠ | User Audit | Audited User | User SYS logged on from EDRSR17P1. | Sep 28, 2004 5:26:16 AM | 0 | Sep 28, 2004 5:26:16 AM |

**Related Alerts**

| Severity | Target Name | Target Type | Category | Name | Message | Alert Triggered | Last Value | Time |
|---|---|---|---|---|---|---|---|---|
| (No alerts) | | | | | | | | |

2. You notice that the recommended action is to run ADDM to get more performance analysis about your system. Click **Additional Advice**.



View Data: Last 24 hours

Wait Class **Application**

**Statistics for Last 24 Hours**

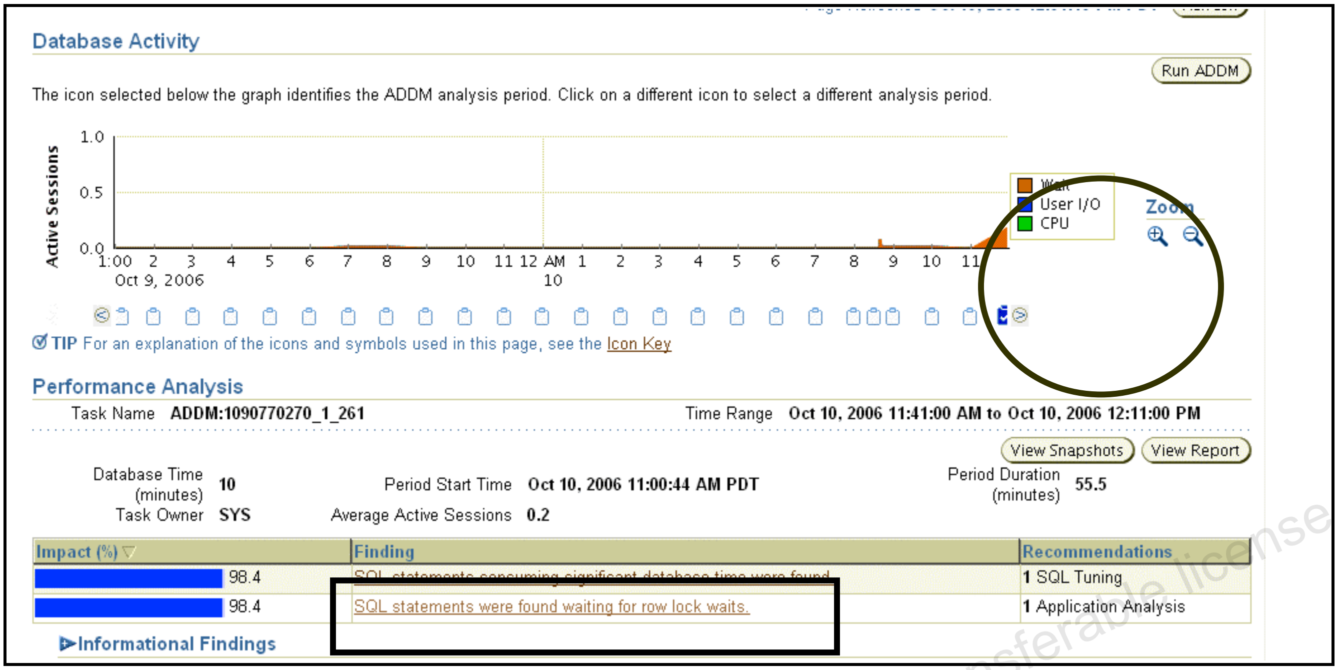
| | |
|---|---|
| Last Known Value | **93.26** |
| Average Value | **18.3** |
| High Value | **100** |
| Low Value | **0.09** |
| □ Warning Threshold | **30** |
| Critical Threshold | **Not Defined** |
| Threshold Occurrences | **1** |

**Recommendations**

Severity ⚠ **Warning**

Recommended Action: Run ADDM to get more performance analysis about your system.

3. Make sure the snapshot you took is selected from the list. Notice that the SQL statements were found waiting for row lock waits, impacting your system for a large part of your database time. **Click this finding** in the list.



Database Activity

The icon selected below the graph identifies the ADDM analysis period. Click on a different icon to select a different analysis period.

TIP For an explanation of the icons and symbols used in this page, see the Icon Key

Performance Analysis

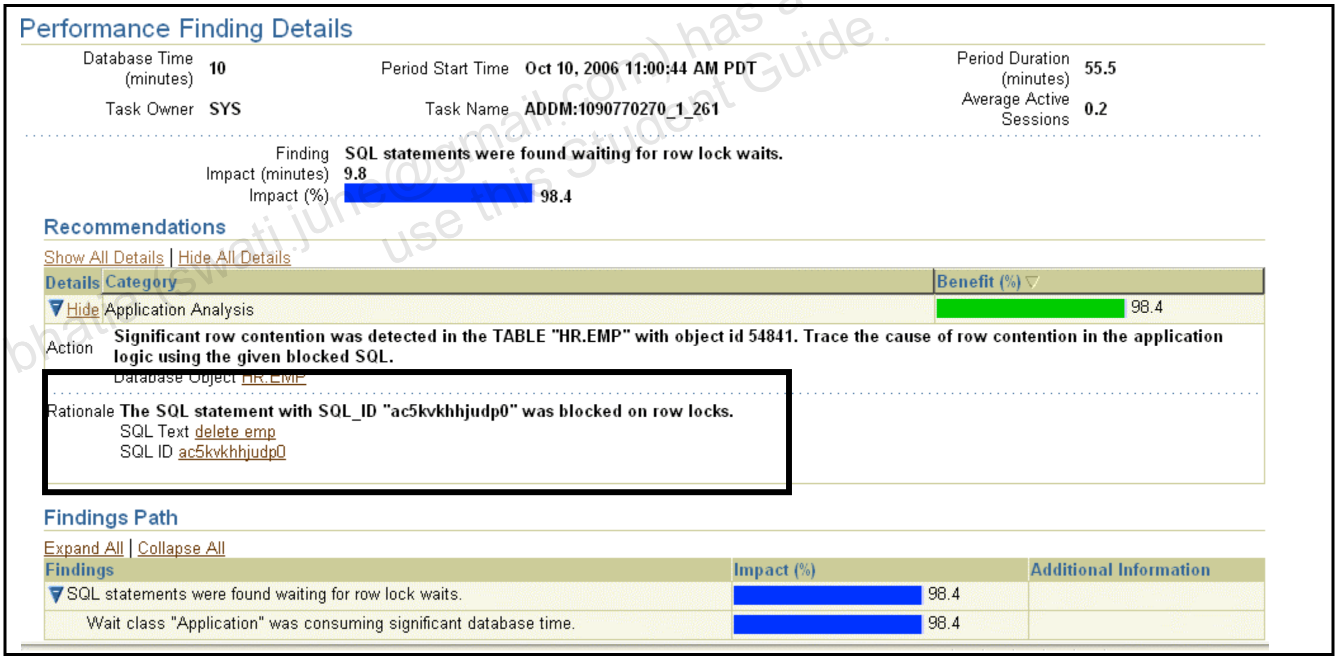
Task Name **ADDM:1090770270_1_261** Time Range **Oct 10, 2006 11:41:00 AM to Oct 10, 2006 12:11:00 PM**

| | | | | | |
|---|---|---|---|---|---|
| Database Time (minutes) | **10** | Period Start Time | **Oct 10, 2006 11:00:44 AM PDT** | Period Duration (minutes) | **55.5** |
| Task Owner | **SYS** | Average Active Sessions | **0.2** | | |

| Impact (%) | Finding | Recommendations |
|---|---|---|
| 98.4 | SQL statements consuming significant database time were found | **1** SQL Tuning |
| 98.4 | SQL statements were found waiting for row lock waits. | **1** Application Analysis |

Informational Findings

4. You see the action that needs to be taken to resolve the performance issue. Look at the rationale displayed.



Performance Finding Details

| | | | | | |
|---|---|---|---|---|---|
| Database Time (minutes) | **10** | Period Start Time | **Oct 10, 2006 11:00:44 AM PDT** | Period Duration (minutes) | **55.5** |
| Task Owner | **SYS** | Task Name | **ADDM:1090770270_1_261** | Average Active Sessions | **0.2** |

Finding **SQL statements were found waiting for row lock waits.**
Impact (minutes) **9.8**
Impact (%) **98.4**

Recommendations

Show All Details | Hide All Details

| Details | Category | Benefit (%) |
|---|---|---|
| Hide | Application Analysis | 98.4 |

Action **Significant row contention was detected in the TABLE "HR.EMP" with object id 54841. Trace the cause of row contention in the application logic using the given blocked SQL.**
Database Object HR.EMP

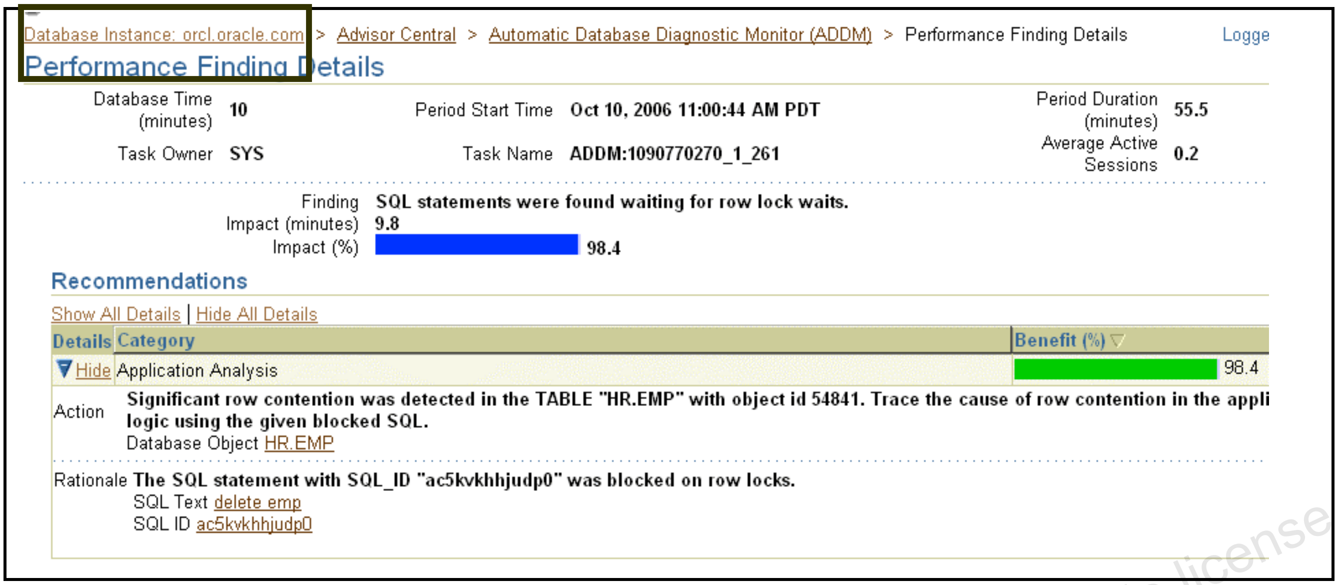
Rationale **The SQL statement with SQL_ID "ac5kvkhhjudp0" was blocked on row locks.**
SQL Text delete emp
SQL ID ac5kvkhhjudp0

Findings Path

Expand All | Collapse All

| Findings | Impact (%) | Additional Information |
|---|---|---|
| SQL statements were found waiting for row lock waits. | 98.4 | |
| Wait class "Application" was consuming significant database time. | 98.4 | |

5. You see the particular SQL ID that is causing the problem. To resolve the performance finding, click the **Database** breadcrumb.



Database Instance: orcl.oracle.com > Advisor Central > Automatic Database Diagnostic Monitor (ADDM) > Performance Finding Details Logge

## Performance Finding Details

| | | | | | |
|---|---|---|---|---|---|
| Database Time (minutes) | 10 | Period Start Time | Oct 10, 2006 11:00:44 AM PDT | Period Duration (minutes) | 55.5 |
| Task Owner | SYS | Task Name | ADDM:1090770270_1_261 | Average Active Sessions | 0.2 |

Finding SQL statements were found waiting for row lock waits.
Impact (minutes) 9.8
Impact (%) 98.4

### Recommendations

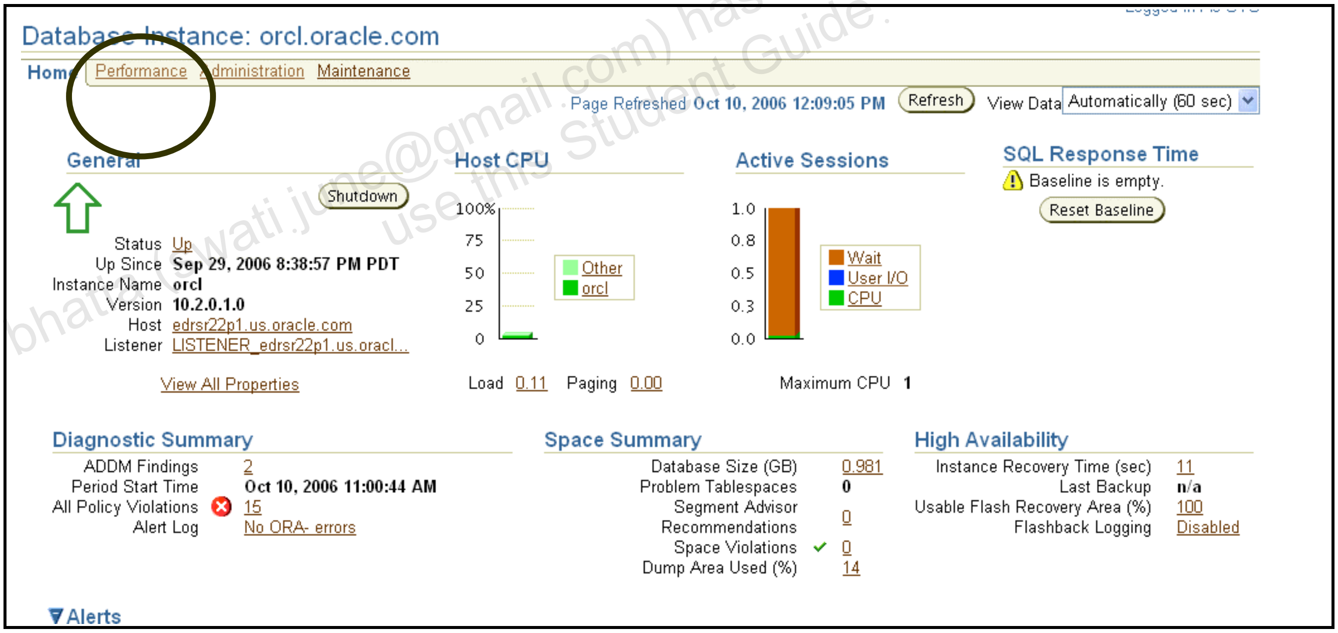
Show All Details | Hide All Details

| Details | Category | Benefit (%) ▽ |
|---|---|---|
| ▼ Hide | Application Analysis | 98.4 |

Action Significant row contention was detected in the TABLE "HR.EMP" with object id 54841. Trace the cause of row contention in the appli logic using the given blocked SQL.
Database Object HR.EMP

Rationale The SQL statement with SQL_ID "ac5kvkhhjudp0" was blocked on row locks.
SQL Text delete emp
SQL ID ac5kvkhhjudp0

6. Click the **Performance** tab.



Database Instance: orcl.oracle.com

Home
Performance
Administration
Maintenance
Page Refreshed Oct 10, 2006 12:09:05 PM
Refresh
View Data Automatically (60 sec)

**General**

Shutdown

Status Up
Up Since Sep 29, 2006 8:38:57 PM PDT
Instance Name orcl
Version 10.2.0.1.0
Host edrsr22p1.us.oracle.com
Listener LISTENER_edrsr22p1.us.oracl...

View All Properties

**Diagnostic Summary**

ADDM Findings 2
Period Start Time Oct 10, 2006 11:00:44 AM
All Policy Violations 15
Alert Log No ORA- errors

**Space Summary**

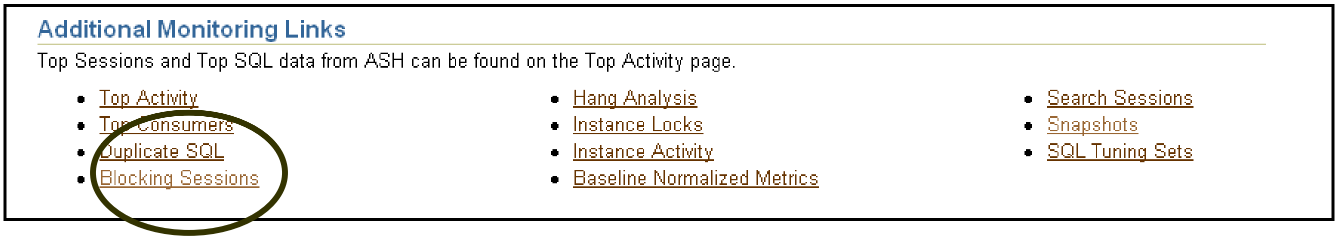
Database Size (GB) 0.981
Problem Tablespaces 0
Segment Advisor Recommendations 0
Space Violations ✓ 0
Dump Area Used (%) 14

**High Availability**

Instance Recovery Time (sec) 11
Last Backup n/a
Usable Flash Recovery Area (%) 100
Flashback Logging Disabled

▼Alerts

7. Scroll down and select **Blocking Sessions** under **Additional Monitoring Links**.

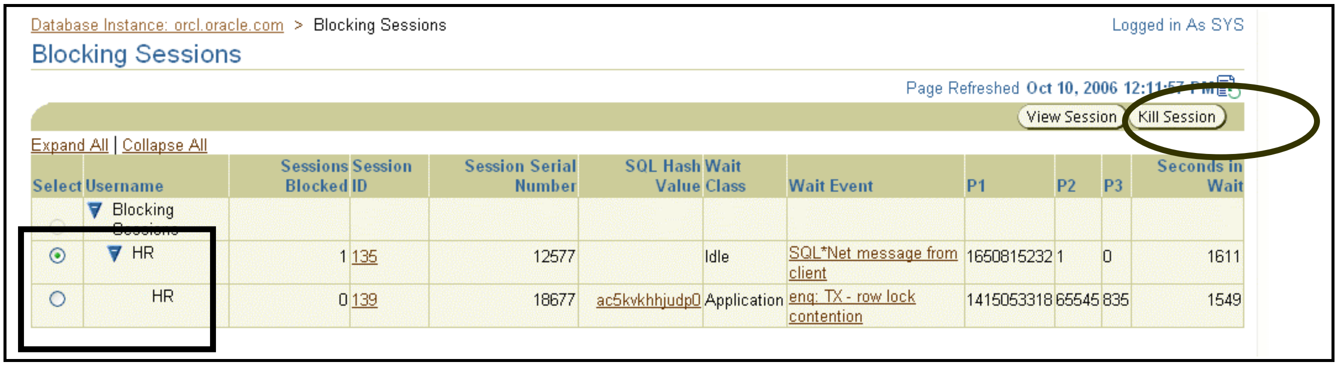


**Additional Monitoring Links**

Top Sessions and Top SQL data from ASH can be found on the Top Activity page.

- Top Activity
- Top Consumers
- Duplicate SQL
- Blocking Sessions
- Hang Analysis
- Instance Locks
- Instance Activity
- Baseline Normalized Metrics
- Search Sessions
- Snapshots
- SQL Tuning Sets

8. Make sure that the highest-level HR is selected, and then click the **Kill Session** button.



Database Instance: orcl.oracle.com > Blocking Sessions

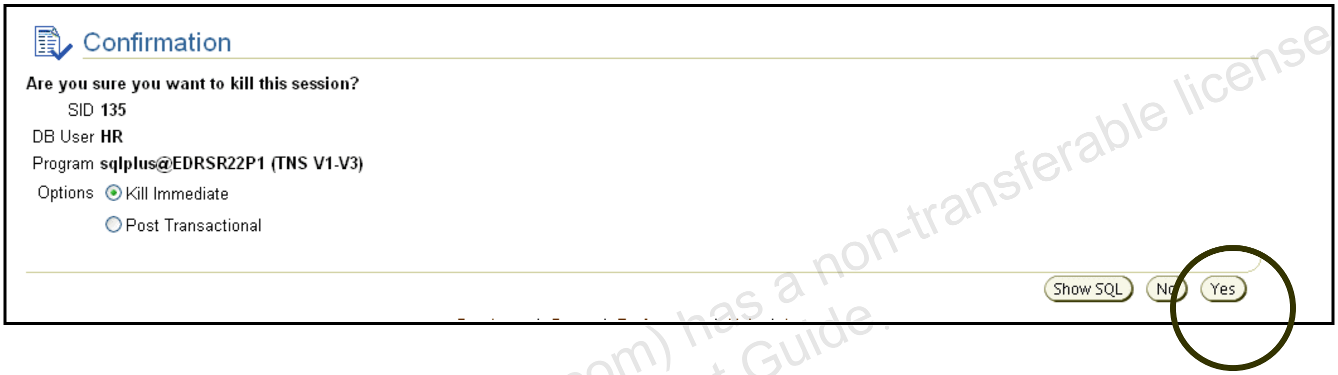
Logged in As SYS

Blocking Sessions

Page Refreshed Oct 10, 2006 12:11:57 PM

| Select | Username | Sessions Blocked | Session ID | Session Serial Number | SQL Hash Value | Wait Class | Wait Event | P1 | P2 | P3 | Seconds in Wait |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ○ | ▼ Blocking Sessions | | | | | | | | | | |
| ● | ▼ HR | 1 | 135 | 12577 | | Idle | SQL*Net message from client | 1650815232 | 1 | 0 | 1611 |
| ○ | HR | 0 | 139 | 18677 | ac5kvkhhjudp0 | Application | enq: TX - row lock contention | 1415053318 | 65545 | 835 | 1549 |

9. Click **Yes** to kill the session.



Confirmation

Are you sure you want to kill this session?

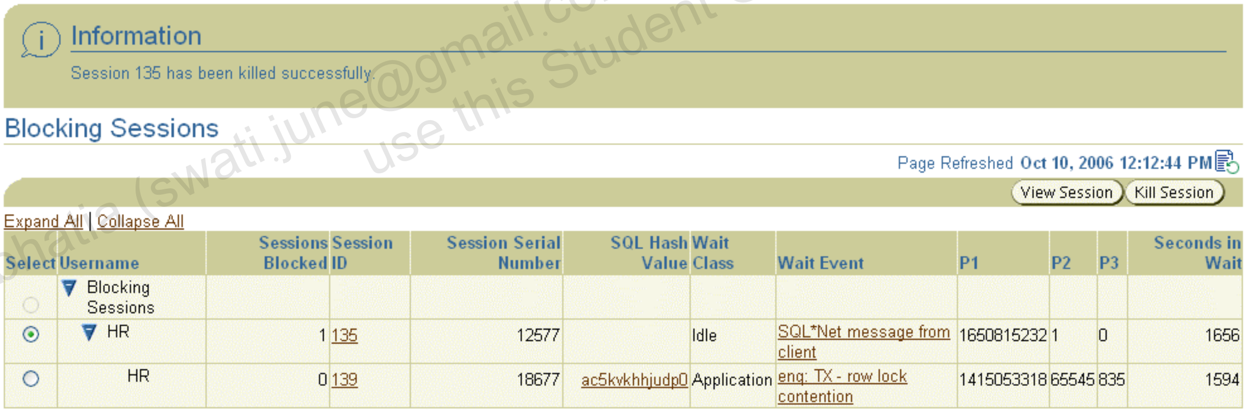
SID **135**

DB User **HR**

Program **sqlplus@EDRSR22P1 (TNS V1-V3)**

Options ● Kill Immediate

○ Post Transactional



Information

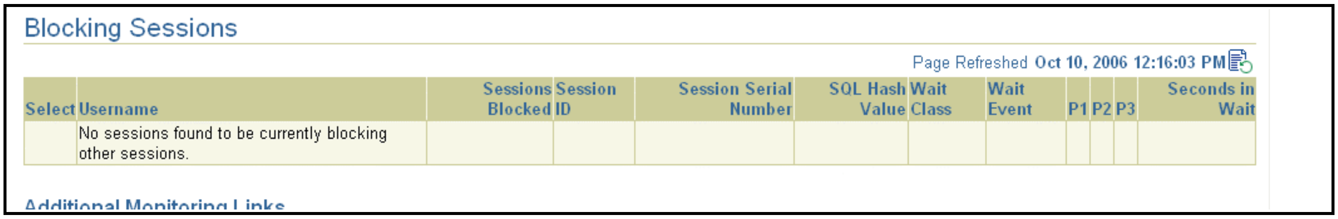
Session 135 has been killed successfully.

Blocking Sessions

Page Refreshed Oct 10, 2006 12:12:44 PM

| Select | Username | Sessions Blocked | Session ID | Session Serial Number | SQL Hash Value | Wait Class | Wait Event | P1 | P2 | P3 | Seconds in Wait |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ○ | ▼ Blocking Sessions | | | | | | | | | | |
| ● | ▼ HR | 1 | 135 | 12577 | | Idle | SQL*Net message from client | 1650815232 | 1 | 0 | 1656 |
| ○ | HR | 0 | 139 | 18677 | ac5kvkhhjudp0 | Application | enq: TX - row lock contention | 1415053318 | 65545 | 835 | 1594 |

10. The session has been killed. Click the **Database** breadcrumb and then click the **Performance** tab. Click **Blocking Sessions** under **Additional Monitoring Links**



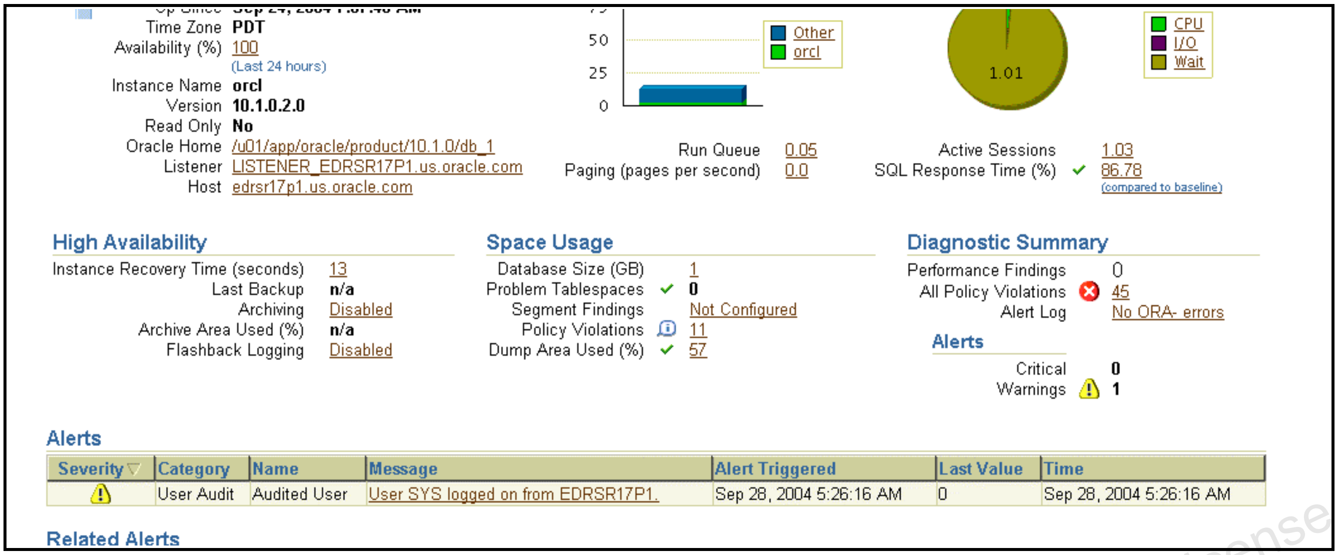
Blocking Sessions

Page Refreshed Oct 10, 2006 12:16:03 PM

| Select | Username | Sessions Blocked | Session ID | Session Serial Number | SQL Hash Value | Wait Class | Wait Event | P1 | P2 | P3 | Seconds in Wait |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | No sessions found to be currently blocking other sessions. | | | | | | | | | | |

Additional Monitoring Links

11. Give a few seconds and perform a manual refresh. Go to the **Home** page.



| | |
|---|---|
| Up Since | **Sep 24, 2004 1:07:48 AM** |
| Time Zone | **PDT** |
| Availability (%) | 100<br>(Last 24 hours) |
| Instance Name | **orcl** |
| Version | **10.1.0.2.0** |
| Read Only | **No** |
| Oracle Home | /u01/app/oracle/product/10.1.0/db_1 |
| Listener | LISTENER_EDRSR17P1.us.oracle.com |
| Host | edrsr17p1.us.oracle.com |

| | |
|---|---|
| Run Queue | 0.05 |
| Paging (pages per second) | 0.0 |

| | | |
|---|---|---|
| Active Sessions | | 1.03 |
| SQL Response Time (%) | ✓ | 86.78<br>(compared to baseline) |

**High Availability**

| | |
|---|---|
| Instance Recovery Time (seconds) | 13 |
| Last Backup | **n/a** |
| Archiving | Disabled |
| Archive Area Used (%) | **n/a** |
| Flashback Logging | Disabled |

**Space Usage**

| | | |
|---|---|---|
| Database Size (GB) | | 1 |
| Problem Tablespaces | ✓ | **0** |
| Segment Findings | | Not Configured |
| Policy Violations | ⓘ | 11 |
| Dump Area Used (%) | ✓ | 57 |

**Diagnostic Summary**

| | | |
|---|---|---|
| Performance Findings | | 0 |
| All Policy Violations | ✖ | 45 |
| Alert Log | | No ORA- errors |

**Alerts**

| | | |
|---|---|---|
| Critical | | **0** |
| Warnings | ⚠ | **1** |

**Alerts**

| Severity ▽ | Category | Name | Message | Alert Triggered | Last Value | Time |
|---|---|---|---|---|---|---|
| ⚠ | User Audit | Audited User | User SYS logged on from EDRSR17P1. | Sep 28, 2004 5:26:16 AM | 0 | Sep 28, 2004 5:26:16 AM |

**Related Alerts**

Note that the alert has disappeared.

12. Close your browser.

## Workshop 2

## Part B: Proactively Tuning Your Database Using the SQL Tuning Advisor

1. Open a command-line window. Change to the **workshop2** subdirectory **under workshop directory** and run the following OS script:

```
$cd workshop/workshop2
$./setup_perflab.sh
```

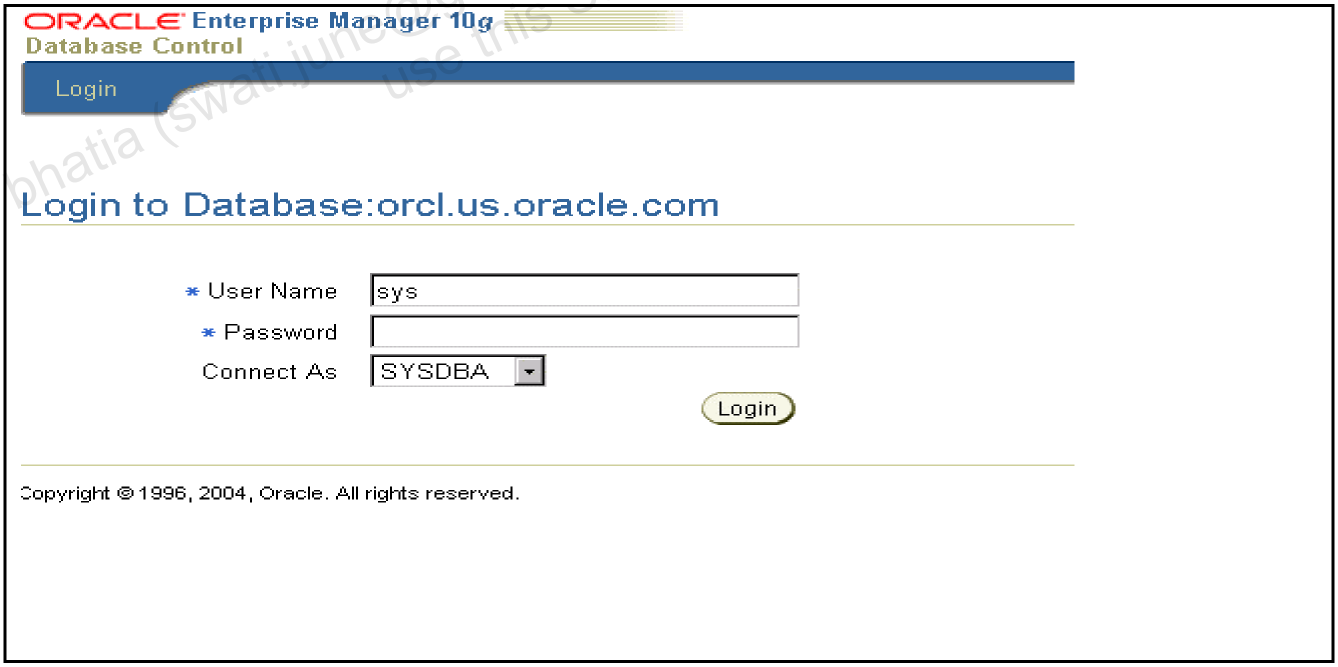
**Note:** This script will take approximately four minutes to run.

**Note:** This script causes the number of user sessions to exponentially increase and use up resources. **If you run this script, but for any reason do not complete the workshop immediately, please run the cleanup script `cleanup_perflab.sh` as instructed at the end of this workshop (`$./cleanup_perflab.sh`)**. When you complete the entire workshop as instructed, your machine will return to optimum performance but it is still recommended to run the cleanup script at the end so that the other labs and workshops are not affected.

2. Open a browser and enter the following URL:

```
http://<hostname>:1158/em
```

Specify sys as the username and the password. Choose SYSDBA from the Connect As drop-down list, then click **Login**.



ORACLE Enterprise Manager 10g
Database Control

Login

Login to Database:orcl.us.oracle.com

* User Name: sys

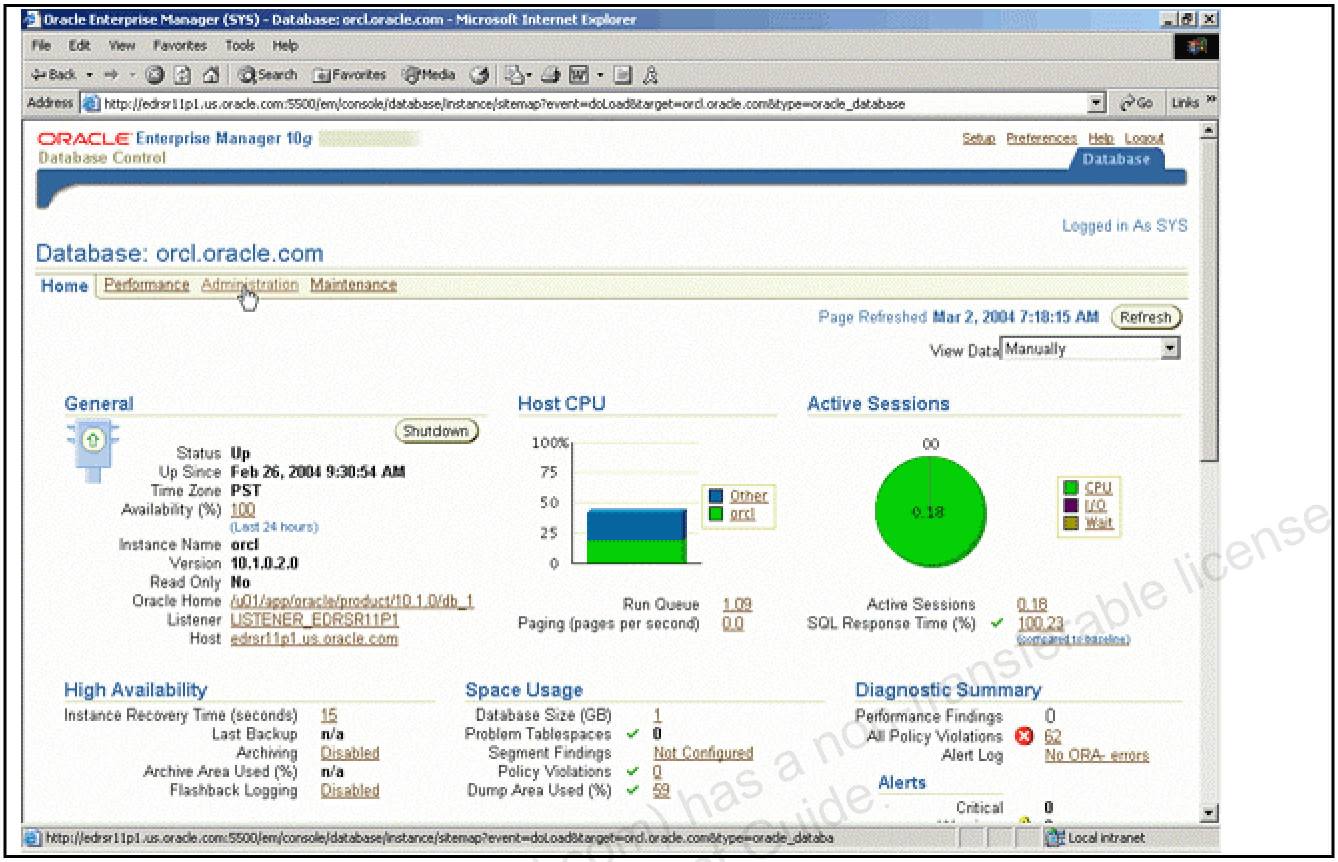
* Password:

Connect As: SYSDBA

Login

Copyright © 1996, 2004, Oracle. All rights reserved.

3. Select the **Administration** link.


Oracle Enterprise Manager (SYS) - Database: orcl.oracle.com - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://edrsr11p1.us.oracle.com:5500/em/console/database/instance/sitemap?event=doLoad&target=orcl.oracle.com&type=oracle_database
Go Links
ORACLE Enterprise Manager 10g
Database Control
Setup Preferences Help Logout
Database
Logged in As SYS
Home Performance Administration Maintenance
Refresh
Shutdown
Other
orcl
CPU
I/O
Wait
100%
75
50
25
0
0.18
http://edrsr11p1.us.oracle.com:5500/em/console/database/instance/sitemap?event=doLoad&target=orcl.oracle.com&type=oracle_databa
Local intranet

Database: orcl.oracle.com

Page Refreshed **Mar 2, 2004 7:18:15 AM**

View Data Manually

General

Status **Up**
Up Since **Feb 26, 2004 9:30:54 AM**
Time Zone **PST**
Availability (%) 100
(Last 24 hours)
Instance Name **orcl**
Version **10.1.0.2.0**
Read Only **No**
Oracle Home /u01/app/oracle/product/10.1.0/db_1
Listener LISTENER_EDRSR11P1
Host edrsr11p1.us.oracle.com

Host CPU

Run Queue 1.09
Paging (pages per second) 0.0

Active Sessions

Active Sessions 0.18
SQL Response Time (%) ✓ 100.23
(compared to baseline)

High Availability

Instance Recovery Time (seconds) 15
Last Backup n/a
Archiving Disabled
Archive Area Used (%) n/a
Flashback Logging Disabled

Space Usage

Database Size (GB) 1
Problem Tablespaces ✓ 0
Segment Findings Not Configured
Policy Violations ✓ 0
Dump Area Used (%) ✓ 59

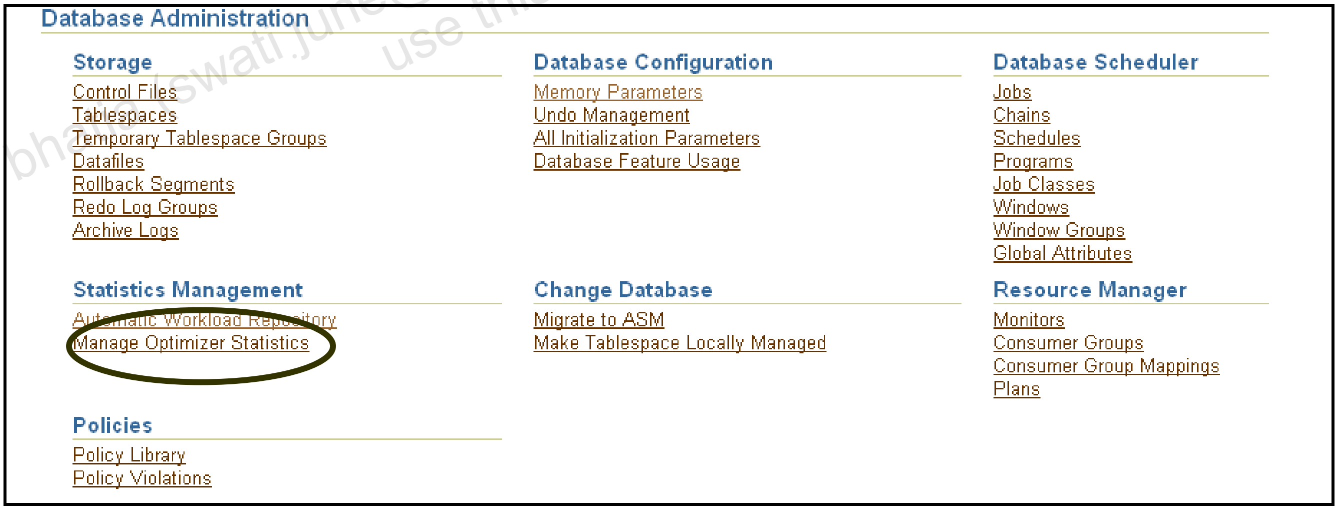
Diagnostic Summary

Performance Findings 0
All Policy Violations ☒ 62
Alert Log No ORA- errors

Alerts

Critical 0

4. In the section titled **Statistics Management**, click the **Automatic Workload Repository** link.



Database Administration

Storage
Control Files
Tablespaces
Temporary Tablespace Groups
Datafiles
Rollback Segments
Redo Log Groups
Archive Logs

Database Configuration
Memory Parameters
Undo Management
All Initialization Parameters
Database Feature Usage

Database Scheduler
Jobs
Chains
Schedules
Programs
Job Classes
Windows
Window Groups
Global Attributes

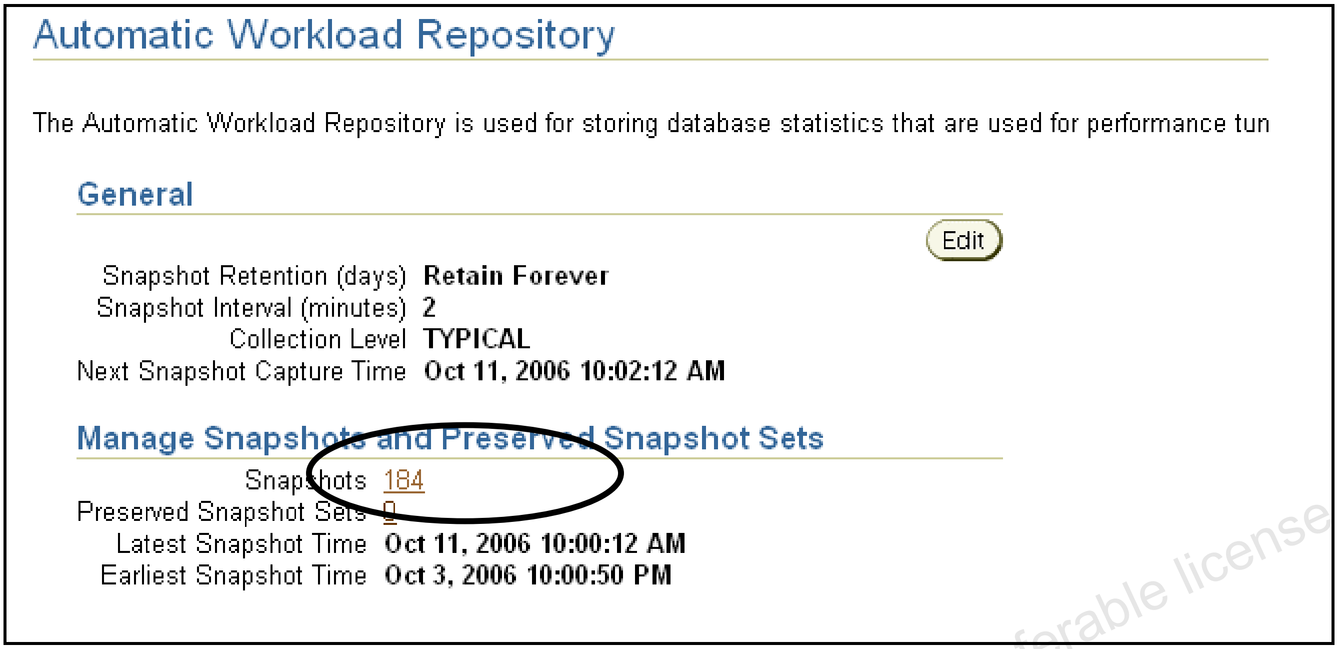
Statistics Management
Automatic Workload Repository
Manage Optimizer Statistics

Change Database
Migrate to ASM
Make Tablespace Locally Managed

Resource Manager
Monitors
Consumer Groups
Consumer Group Mappings
Plans

Policies
Policy Library
Policy Violations

5. Determine how many snapshots have already been collected for this database. Look under Snapshots for the count and the time the last ADDM snapshot was taken. There should be at least three snapshots. Click the **Database** breadcrumb.



## Automatic Workload Repository

The Automatic Workload Repository is used for storing database statistics that are used for performance tun

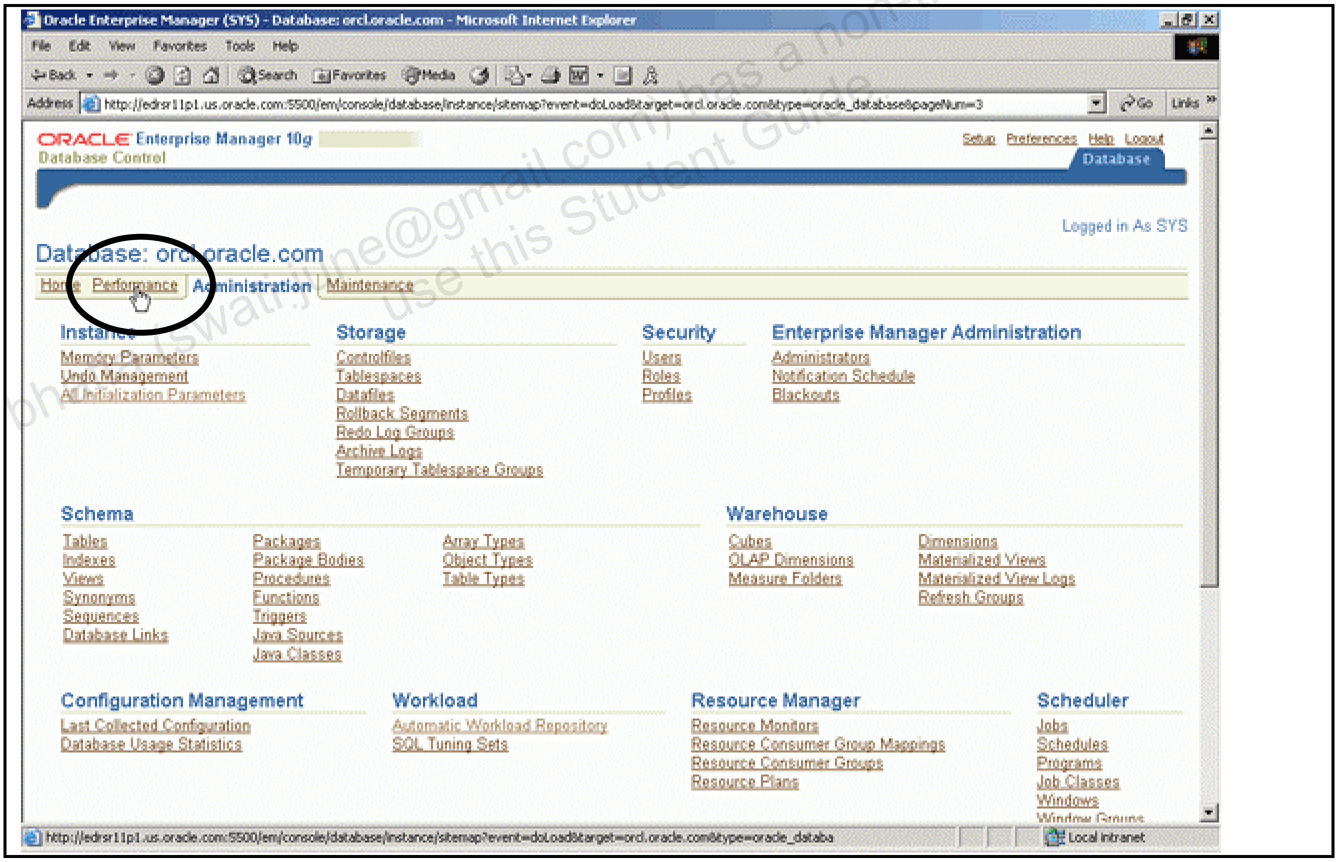
### General

| | |
|---|---|
| Snapshot Retention (days) | **Retain Forever** |
| Snapshot Interval (minutes) | **2** |
| Collection Level | **TYPICAL** |
| Next Snapshot Capture Time | **Oct 11, 2006 10:02:12 AM** |

### Manage Snapshots and Preserved Snapshot Sets

| | |
|---|---|
| Snapshots | 184 |
| Preserved Snapshot Sets | 0 |
| Latest Snapshot Time | **Oct 11, 2006 10:00:12 AM** |
| Earliest Snapshot Time | **Oct 3, 2006 10:00:50 PM** |

6. Click the **Performance** tab.



Oracle Enterprise Manager (SYS) - Database: orcl.oracle.com - Microsoft Internet Explorer

File Edit View Favorites Tools Help
Back Search Favorites Media
Address
Go Links

http://edrsr11p1.us.oracle.com:5500/em/console/database/instance/sitemap?event=doLoad&target=orcl.oracle.com&type=oracle_database&pageNum=3

ORACLE Enterprise Manager 10g
Database Control
Setup Preferences Help Logout
Database
Logged in As SYS

Database: orcl.oracle.com

Home Performance Administration Maintenance

**Instance**: Memory Parameters, Undo Management, All Initialization Parameters

**Storage**: Controlfiles, Tablespaces, Datafiles, Rollback Segments, Redo Log Groups, Archive Logs, Temporary Tablespace Groups

**Security**: Users, Roles, Profiles

**Enterprise Manager Administration**: Administrators, Notification Schedule, Blackouts

**Schema**: Tables, Indexes, Views, Synonyms, Sequences, Database Links, Packages, Package Bodies, Procedures, Functions, Triggers, Java Sources, Java Classes, Array Types, Object Types, Table Types

**Warehouse**: Cubes, OLAP Dimensions, Measure Folders, Dimensions, Materialized Views, Materialized View Logs, Refresh Groups

**Configuration Management**: Last Collected Configuration, Database Usage Statistics

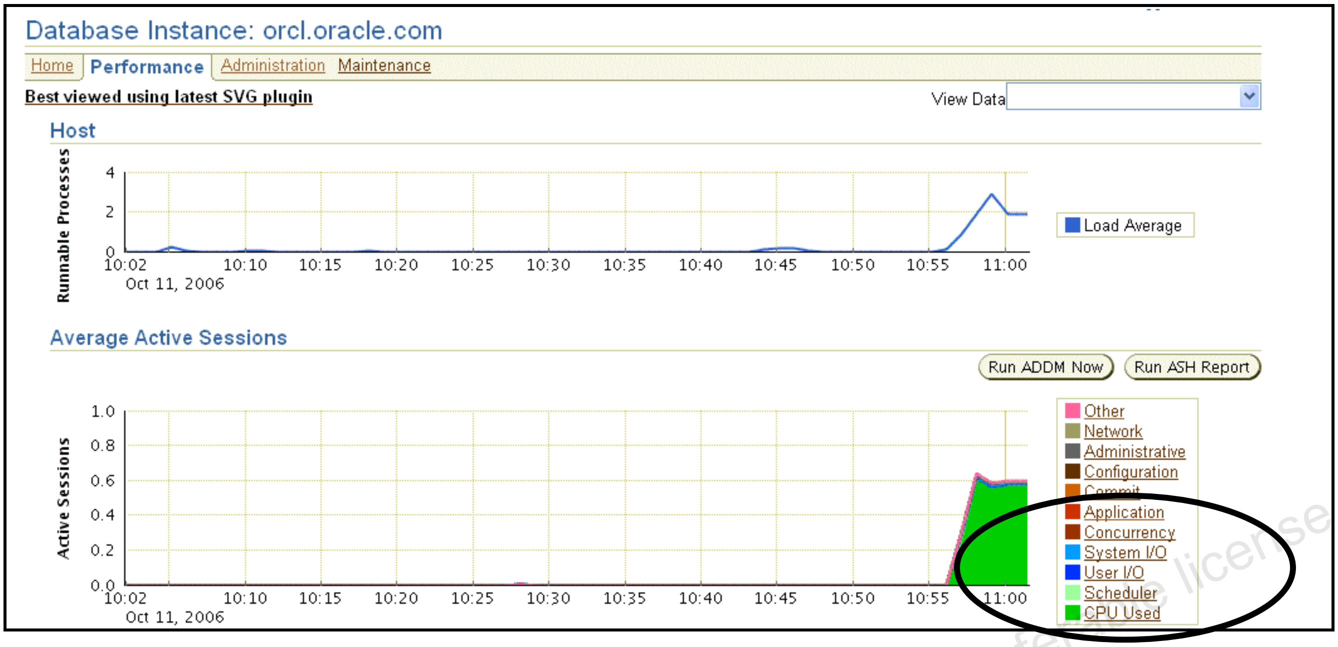
**Workload**: Automatic Workload Repository, SQL Tuning Sets

**Resource Manager**: Resource Monitors, Resource Consumer Group Mappings, Resource Consumer Groups, Resource Plans

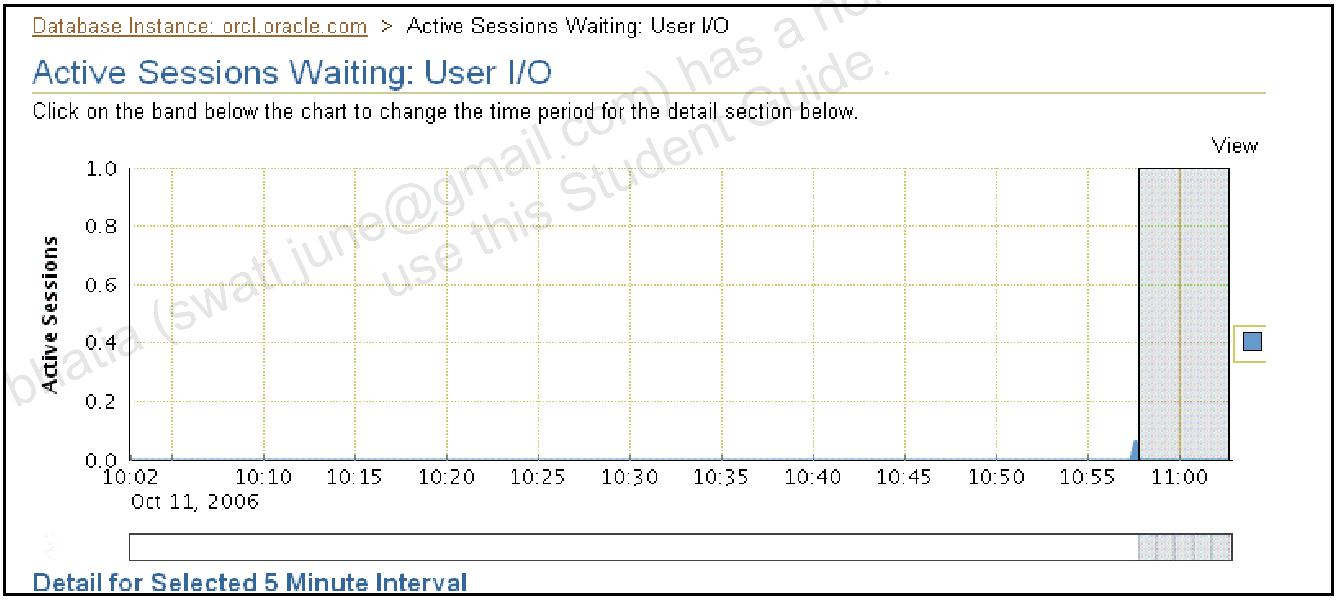
**Scheduler**: Jobs, Schedules, Programs, Job Classes, Windows, Window Groups

http://edrsr11p1.us.oracle.com:5500/em/console/database/instance/sitemap?event=doLoad&target=orcl.oracle.com&type=oracle_databa

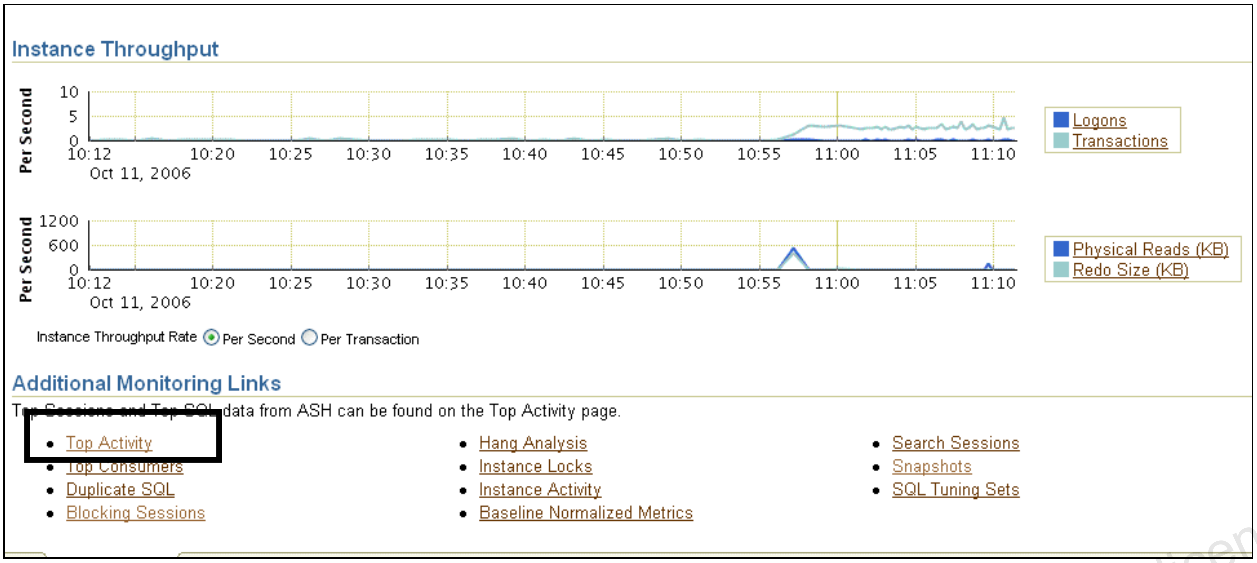
7. In the Performance window, review the **Average active sessions graph**. A chart representing the current workload of your database is shown. (It may take a minute for the chart to be populated with data.) This chart is populated with data collected by the ADDM snapshots. To the side of the graph is the legend. Each legend entry is coded to a different color. Click the **User I/O** link. **Do not** wait for graph to drop.



8. Below the **Active Sessions Waiting: User I/O** chart, under Additional Monitoring Links, click **Top Activity**.



Database Instance: orcl.oracle.com > Active Sessions Waiting: User I/O
Active Sessions Waiting: User I/O
Click on the band below the chart to change the time period for the detail section below.
View
Active Sessions
1.0
0.8
0.6
0.4
0.2
0.0
10:02
Oct 11, 2006
10:10
10:15
10:20
10:25
10:30
10:35
10:40
10:45
10:50
10:55
11:00
Detail for Selected 5 Minute Interval



Instance Throughput

Instance Throughput Rate ● Per Second ○ Per Transaction

Additional Monitoring Links

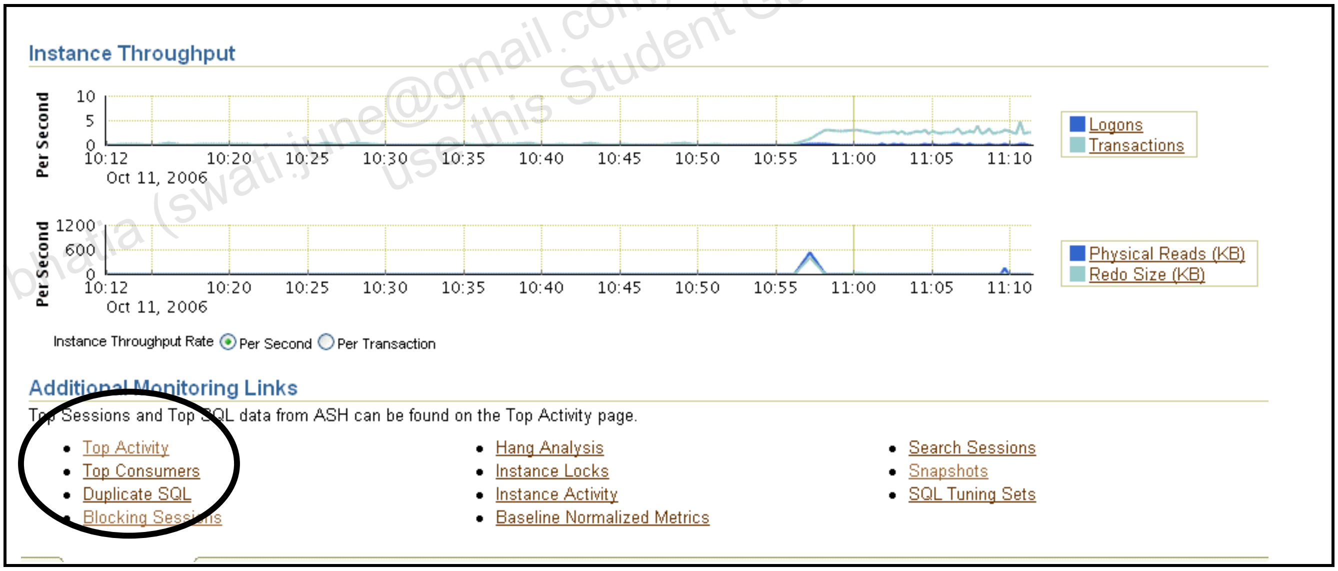
Top Sessions and Top SQL data from ASH can be found on the Top Activity page.

- Top Activity
- Top Consumers
- Duplicate SQL
- Blocking Sessions
- Hang Analysis
- Instance Locks
- Instance Activity
- Baseline Normalized Metrics
- Search Sessions
- Snapshots
- SQL Tuning Sets

### Examining Top SQL for a Database Wait Class

As was shown in the previous task, there is one SQL statement causing the majority of the database wait. In this task, you will drill down to find the root cause. Perform the following steps:

1. From within the **Active Sessions Waiting: User I/O page**, under Additional Monitoring Links, click **Top Activity**.



Instance Throughput

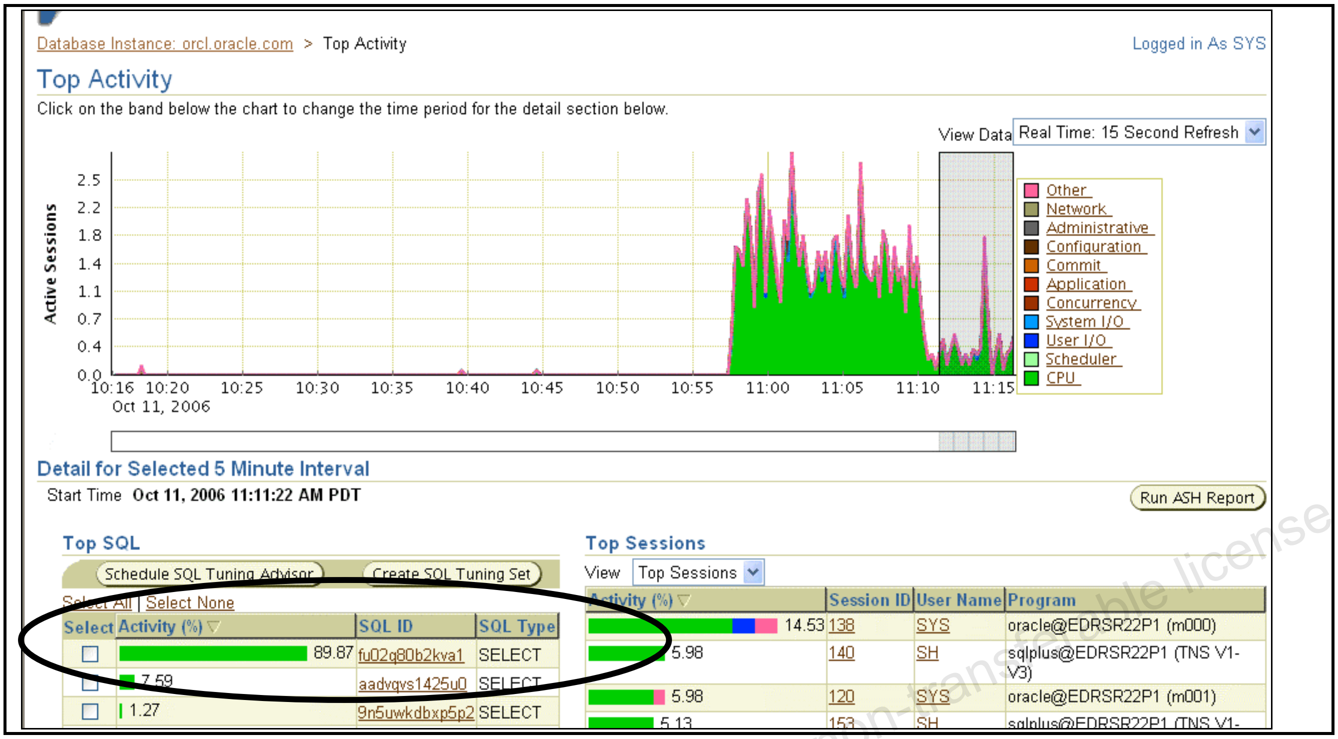
Instance Throughput Rate ● Per Second ○ Per Transaction

Additional Monitoring Links

Top Sessions and Top SQL data from ASH can be found on the Top Activity page.

- Top Activity
- Top Consumers
- Duplicate SQL
- Blocking Sessions
- Hang Analysis
- Instance Locks
- Instance Activity
- Baseline Normalized Metrics
- Search Sessions
- Snapshots
- SQL Tuning Sets

2. On the detail page that appears, view the **Wait Events for Top SQL** table, which is ordered by Activity (%). You can see the **Top SQL statement** spent most of its time on the “db file scattered read” activity. Click the **SQL ID** of the SQL statement with the highest percentage of activity.



Database Instance: orcl.oracle.com > Top Activity

Logged in As SYS

Top Activity

Click on the band below the chart to change the time period for the detail section below.

View Data: Real Time: 15 Second Refresh

Detail for Selected 5 Minute Interval

Start Time **Oct 11, 2006 11:11:22 AM PDT**

Run ASH Report

Top SQL

Schedule SQL Tuning Advisor | Create SQL Tuning Set

Select All | Select None

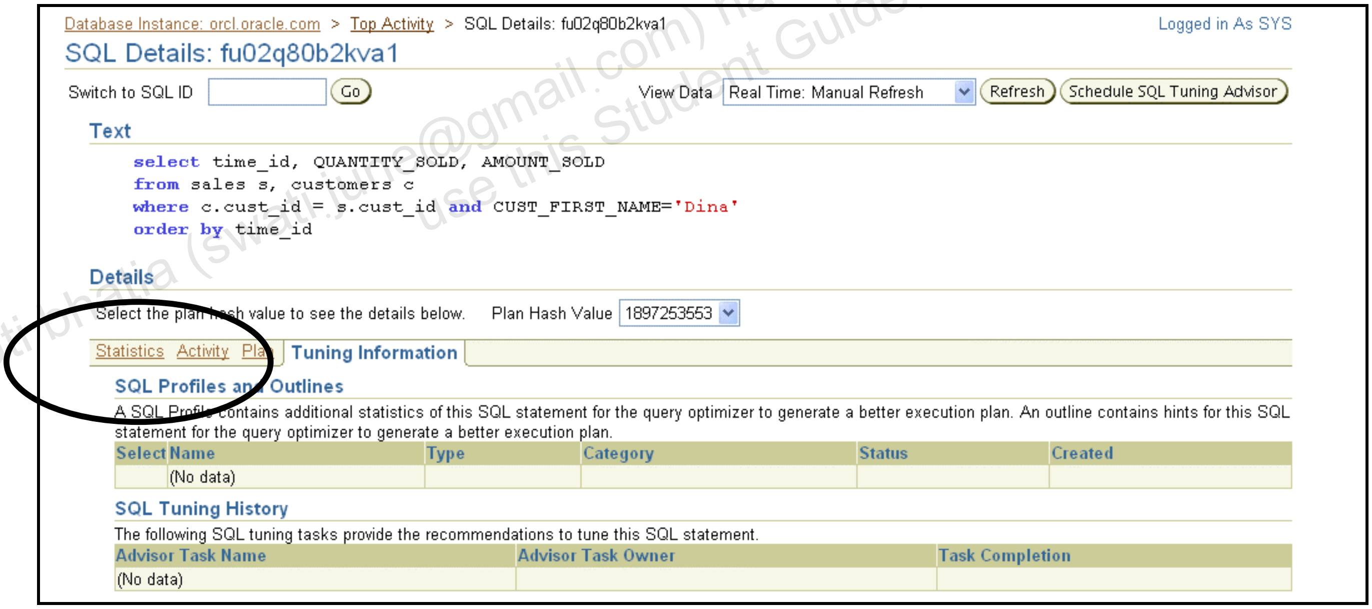
| Select | Activity (%) | SQL ID | SQL Type |
|---|---|---|---|
| □ | 89.87 | fu02q80b2kva1 | SELECT |
| □ | 7.59 | aadvqvs1425u0 | SELECT |
| □ | 1.27 | 9n5uwkdbxp5p2 | SELECT |

Top Sessions

View: Top Sessions

| Activity (%) | Session ID | User Name | Program |
|---|---|---|---|
| 14.53 | 138 | SYS | oracle@EDRSR22P1 (m000) |
| 5.98 | 140 | SH | sqlplus@EDRSR22P1 (TNS V1-V3) |
| 5.98 | 120 | SYS | oracle@EDRSR22P1 (m001) |
| 5.13 | 153 | SH | sqlplus@EDRSR22P1 (TNS V1- |

3. The execution plan for this SQL statement is displayed. Click the **Current Statistics** tab.



Database Instance: orcl.oracle.com > Top Activity > SQL Details: fu02q80b2kva1

Logged in As SYS

SQL Details: fu02q80b2kva1

Switch to SQL ID: Go

View Data: Real Time: Manual Refresh | Refresh | Schedule SQL Tuning Advisor

Text

```
select time_id, QUANTITY_SOLD, AMOUNT_SOLD
from sales s, customers c
where c.cust_id = s.cust_id and CUST_FIRST_NAME='Dina'
order by time_id
```

Details

Select the plan hash value to see the details below. Plan Hash Value: 1897253553

Statistics | Activity | Plan | **Tuning Information**

SQL Profiles and Outlines

A SQL Profile contains additional statistics of this SQL statement for the query optimizer to generate a better execution plan. An outline contains hints for this SQL statement for the query optimizer to generate a better execution plan.

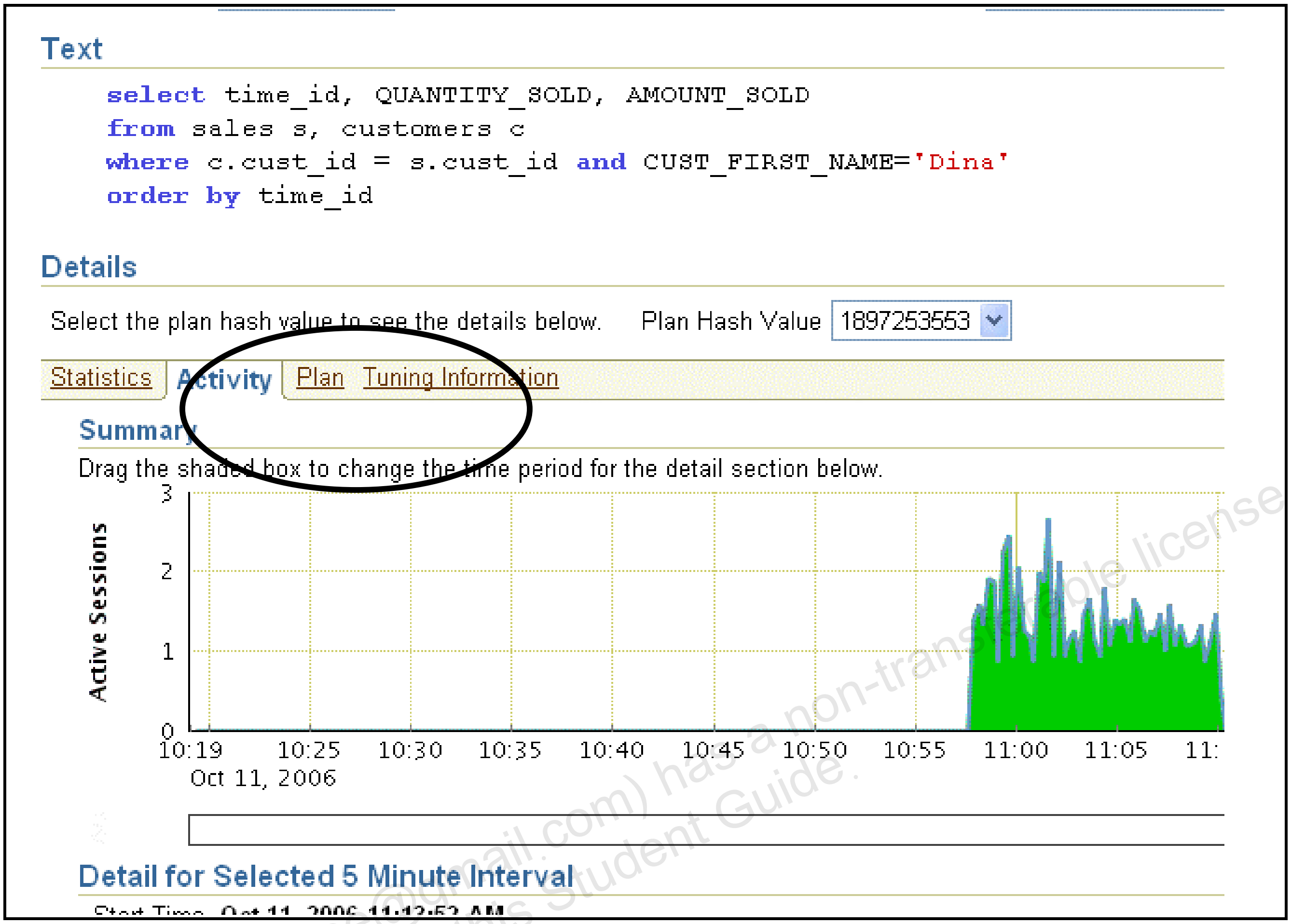
| Select | Name | Type | Category | Status | Created |
|---|---|---|---|---|---|
| | (No data) | | | | |

SQL Tuning History

The following SQL tuning tasks provide the recommendations to tune this SQL statement.

| Advisor Task Name | Advisor Task Owner | Task Completion |
|---|---|---|
| (No data) | | |

4. The activity for this SQL statement is displayed. Click the **Plan** tab.



Text

```
select time_id, QUANTITY_SOLD, AMOUNT_SOLD
from sales s, customers c
where c.cust_id = s.cust_id and CUST_FIRST_NAME='Dina'
order by time_id
```

Details

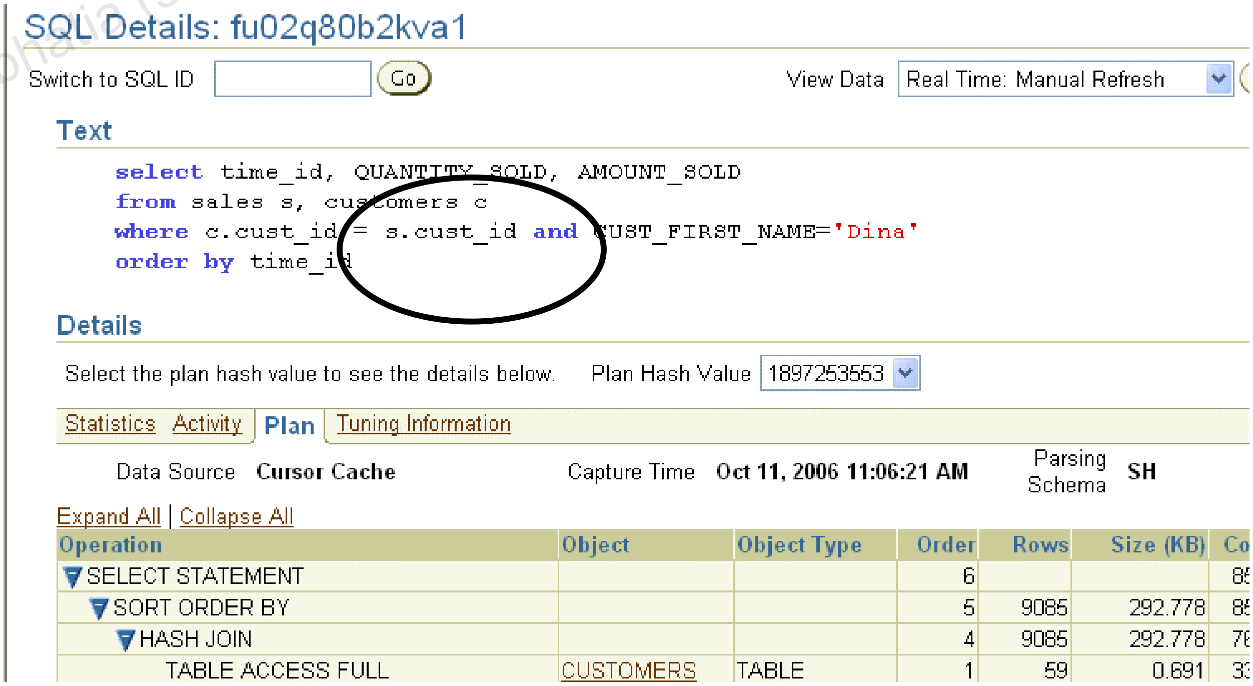
Select the plan hash value to see the details below. Plan Hash Value 1897253553

Drag the shaded box to change the time period for the detail section below.

Detail for Selected 5 Minute Interval

Start Time Oct 11, 2006 11:12:52 AM

5. The execution plan is displayed. Then click the Statistics tab to view the statistics. The statistical analysis chart for this SQL statement is displayed.



SQL Details: fu02q80b2kva1

Text

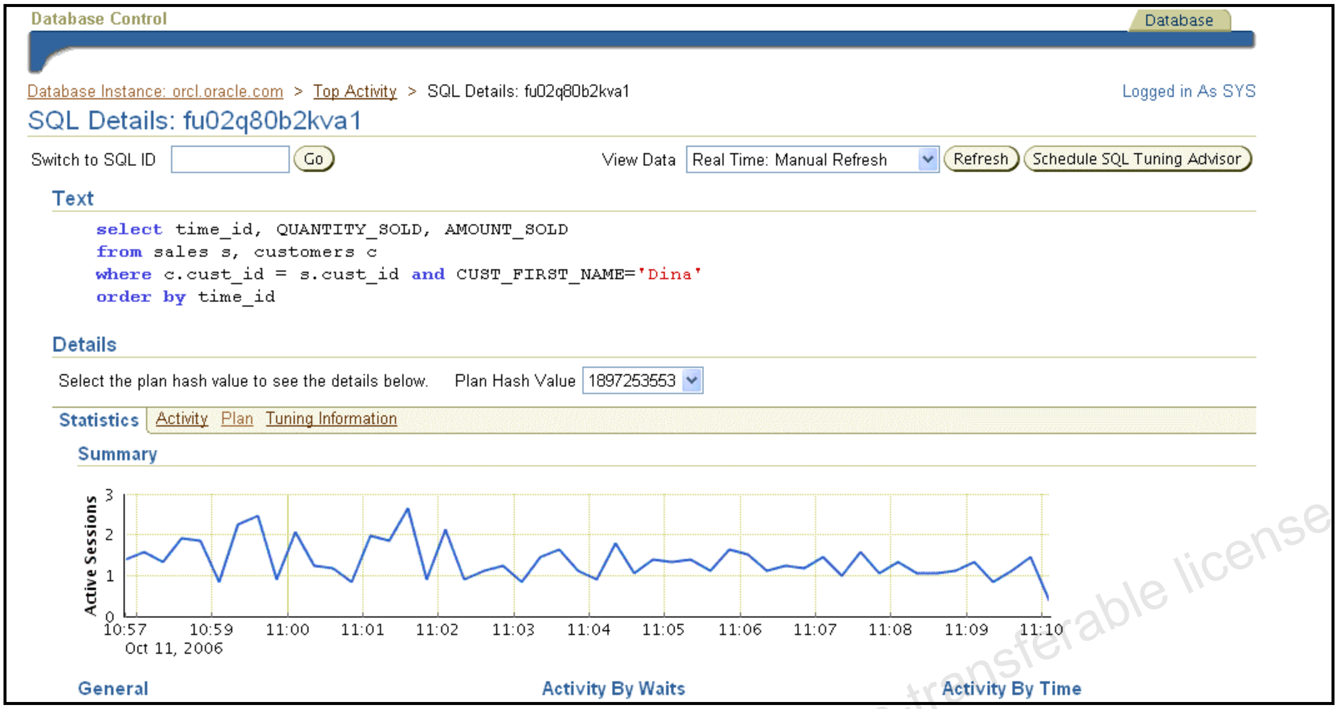
```
select time_id, QUANTITY_SOLD, AMOUNT_SOLD
from sales s, customers c
where c.cust_id = s.cust_id and CUST_FIRST_NAME='Dina'
order by time_id
```

Details

Select the plan hash value to see the details below. Plan Hash Value 1897253553

Data Source **Cursor Cache** Capture Time **Oct 11, 2006 11:06:21 AM** Parsing Schema **SH**

| Operation | Object | Object Type | Order | Rows | Size (KB) | Co |
|---|---|---|---|---|---|---|
| ▼SELECT STATEMENT | | | 6 | | | 85 |
| ▼SORT ORDER BY | | | 5 | 9085 | 292.778 | 85 |
| ▼HASH JOIN | | | 4 | 9085 | 292.778 | 76 |
| TABLE ACCESS FULL | CUSTOMERS | TABLE | 1 | 59 | 0.691 | 33 |


Database Control
Database
Database Instance: orcl.oracle.com > Top Activity > SQL Details: fu02q80b2kva1
Logged in As SYS

SQL Details: fu02q80b2kva1

Text

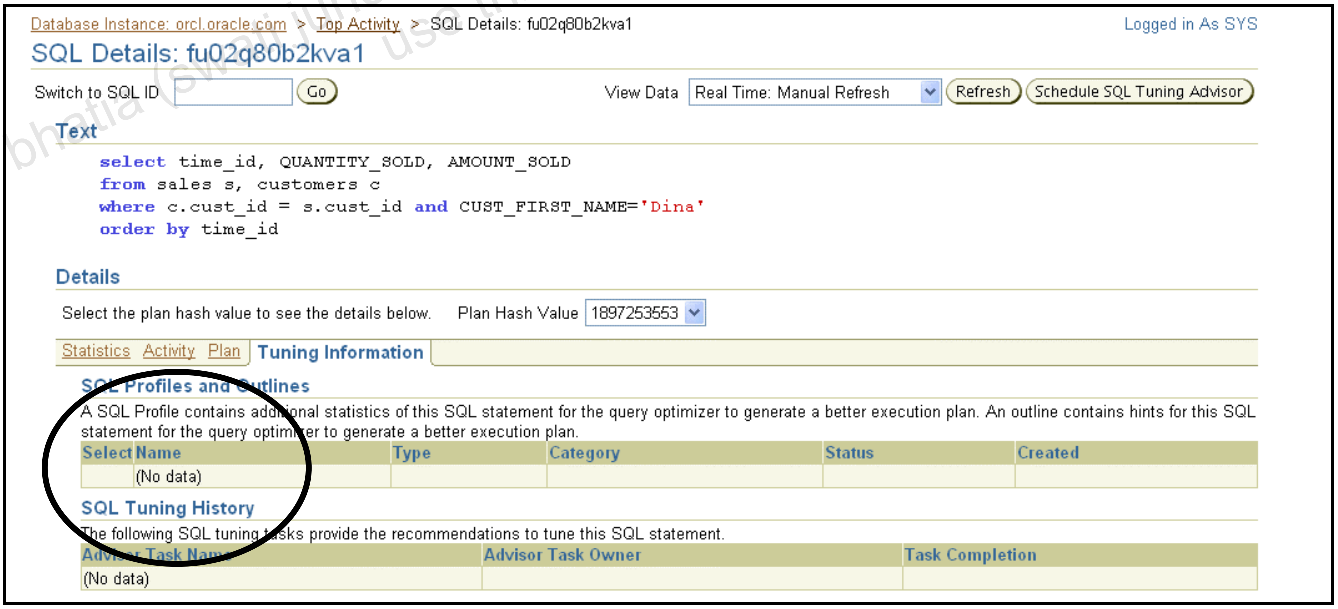
```
select time_id, QUANTITY_SOLD, AMOUNT_SOLD
from sales s, customers c
where c.cust_id = s.cust_id and CUST_FIRST_NAME='Dina'
order by time_id
```

Details

Select the plan hash value to see the details below. Plan Hash Value 1897253553

6. From the displayed charts, it can be determined that CPU resource usage is increasing, and the time it takes to execute this SQL statement is also increasing. Click the **Tuning History** tab.

7. The previous tuning recommendations for this SQL statement are displayed. At this time, there are none. You are now ready to tune the SQL statement by using the **SQL Tuning Advisor**.


Database Instance: orcl.oracle.com > Top Activity > SQL Details: fu02q80b2kva1
Logged in As SYS

SQL Details: fu02q80b2kva1

Switch to SQL ID
Go
View Data
Real Time: Manual Refresh
Refresh
Schedule SQL Tuning Advisor

Text

```
select time_id, QUANTITY_SOLD, AMOUNT_SOLD
from sales s, customers c
where c.cust_id = s.cust_id and CUST_FIRST_NAME='Dina'
order by time_id
```

Details

Select the plan hash value to see the details below. Plan Hash Value 1897253553

Statistics
Activity
Plan
Tuning Information

SQL Profiles and Outlines

A SQL Profile contains additional statistics of this SQL statement for the query optimizer to generate a better execution plan. An outline contains hints for this SQL statement for the query optimizer to generate a better execution plan.

| Select | Name | Type | Category | Status | Created |
|---|---|---|---|---|---|
| | (No data) | | | | |

SQL Tuning History

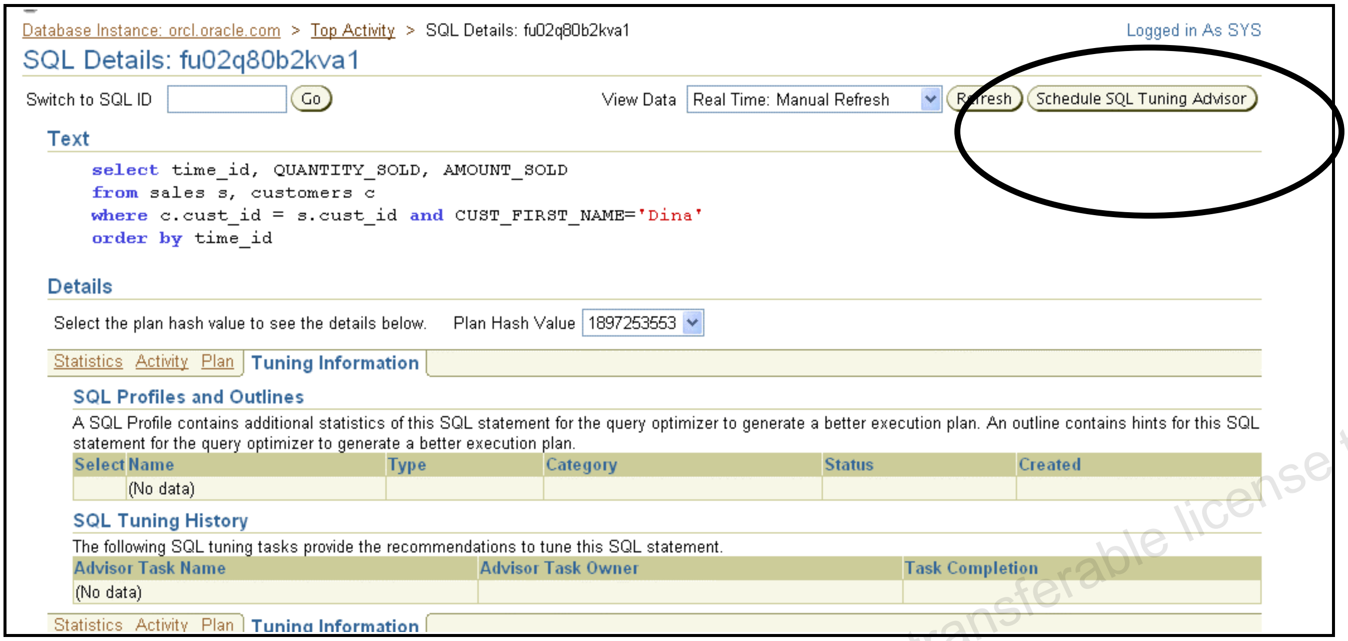
The following SQL tuning tasks provide the recommendations to tune this SQL statement.

| Advisor Task Name | Advisor Task Owner | Task Completion |
|---|---|---|
| (No data) | | |

## Tuning a SQL Statement by Using the SQL Tuning Advisor

As determined in the previous section on reactive tuning, the targeted SQL statement needs tuning. The **SQL Tuning Advisor** will tune the execution plan for you. Perform the following steps:

1. Click **Run SQL Tuning Advisor**.



Database Instance: orcl.oracle.com > Top Activity > SQL Details: fu02q80b2kva1

Logged in As SYS

## SQL Details: fu02q80b2kva1

Switch to SQL ID [ ] Go

View Data: Real Time: Manual Refresh

### Text

```
select time_id, QUANTITY_SOLD, AMOUNT_SOLD
from sales s, customers c
where c.cust_id = s.cust_id and CUST_FIRST_NAME='Dina'
order by time_id
```

### Details

Select the plan hash value to see the details below. Plan Hash Value: 1897253553

#### SQL Profiles and Outlines

A SQL Profile contains additional statistics of this SQL statement for the query optimizer to generate a better execution plan. An outline contains hints for this SQL statement for the query optimizer to generate a better execution plan.

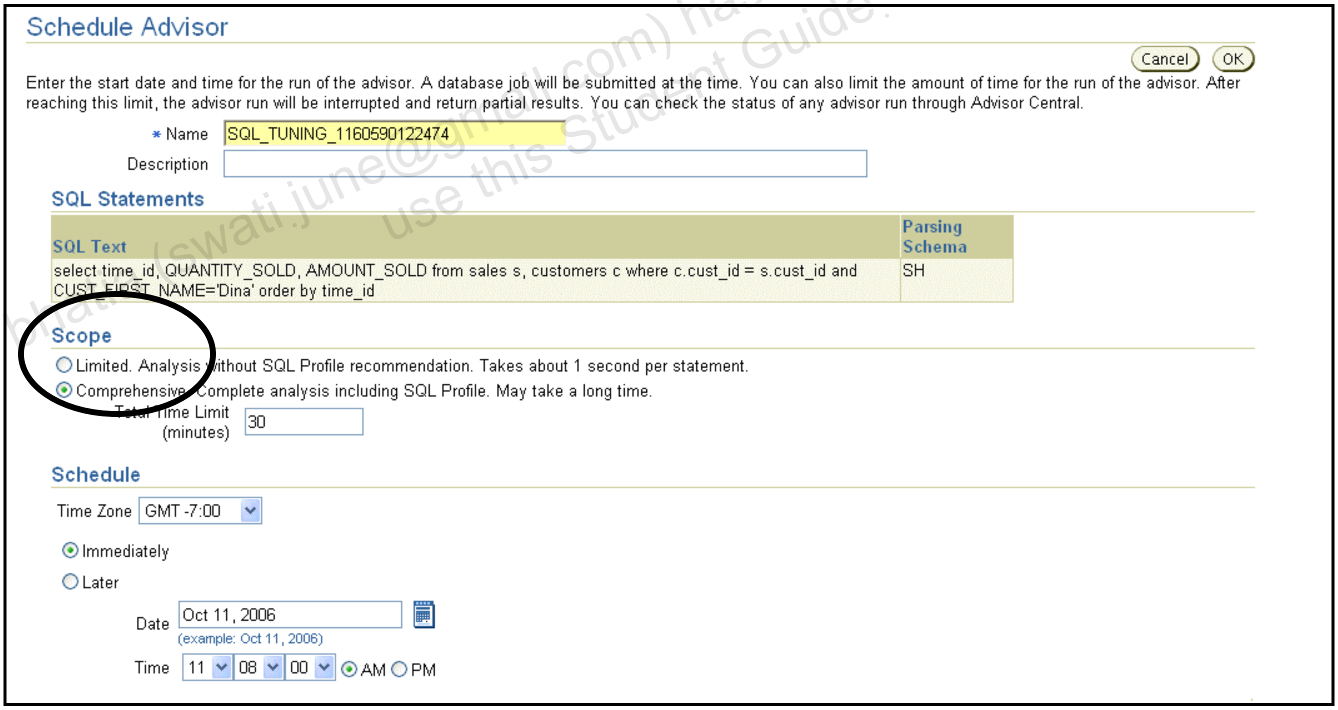
| Select | Name | Type | Category | Status | Created |
|---|---|---|---|---|---|
| | (No data) | | | | |

#### SQL Tuning History

The following SQL tuning tasks provide the recommendations to tune this SQL statement.

| Advisor Task Name | Advisor Task Owner | Task Completion |
|---|---|---|
| (No data) | | |

2. At the **Schedule Advisor** window, make sure that the **Scope Comprehensive** is selected and the job will be scheduled **Immediately**. Click **OK**.



## Schedule Advisor

Enter the start date and time for the run of the advisor. A database job will be submitted at the time. You can also limit the amount of time for the run of the advisor. After reaching this limit, the advisor run will be interrupted and return partial results. You can check the status of any advisor run through Advisor Central.

* Name: SQL_TUNING_1160590122474

Description:

### SQL Statements

| SQL Text | Parsing Schema |
|---|---|
| select time_id, QUANTITY_SOLD, AMOUNT_SOLD from sales s, customers c where c.cust_id = s.cust_id and CUST_FIRST_NAME='Dina' order by time_id | SH |

### Scope

○ Limited. Analysis without SQL Profile recommendation. Takes about 1 second per statement.

● Comprehensive. Complete analysis including SQL Profile. May take a long time.

Total Time Limit (minutes): 30

### Schedule

Time Zone: GMT -7:00

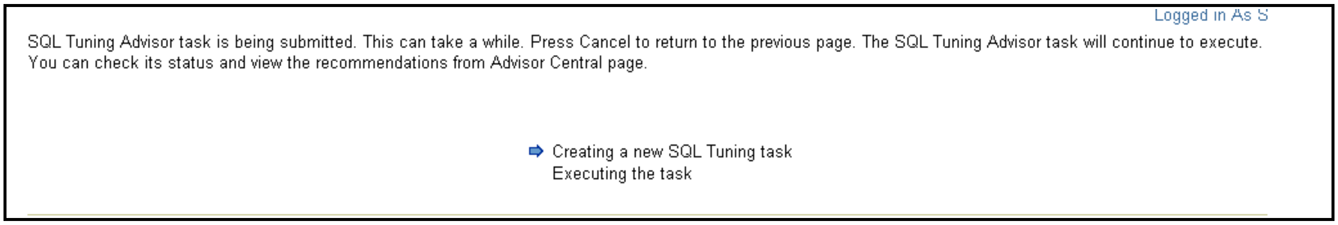
● Immediately

○ Later

Date: Oct 11, 2006 (example: Oct 11, 2006)

Time: 11 : 08 : 00 ● AM ○ PM

3. The **SQL Tuning Advisor** will create a task to analyze the SQL statement and, on completion of this task, will display a set of tuning recommendations.

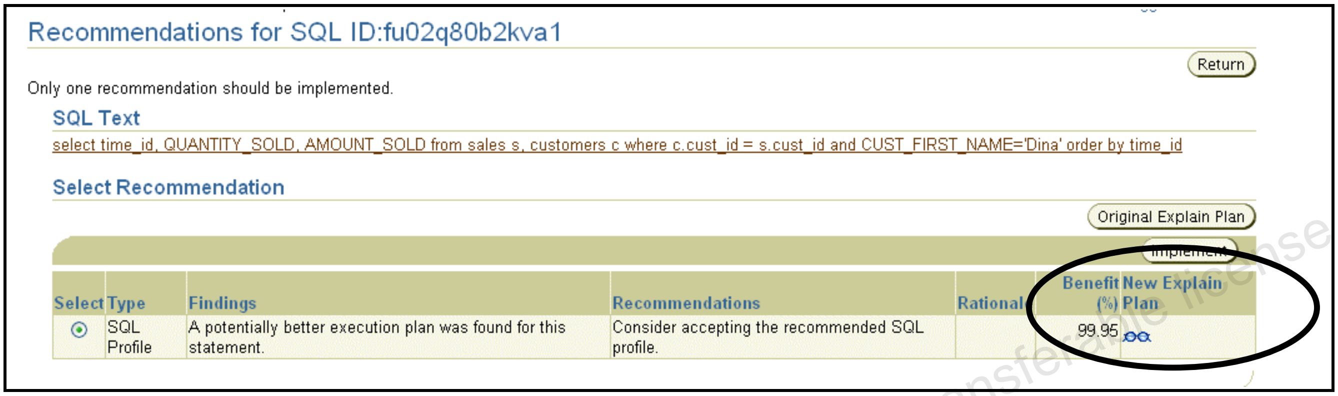


Logged in As S

SQL Tuning Advisor task is being submitted. This can take a while. Press Cancel to return to the previous page. The SQL Tuning Advisor task will continue to execute. You can check its status and view the recommendations from Advisor Central page.

➡ Creating a new SQL Tuning task
Executing the task

4. Click the **New Explain Plan** eyeglass icon to view the suggested change.



Recommendations for SQL ID:fu02q80b2kva1

Only one recommendation should be implemented.

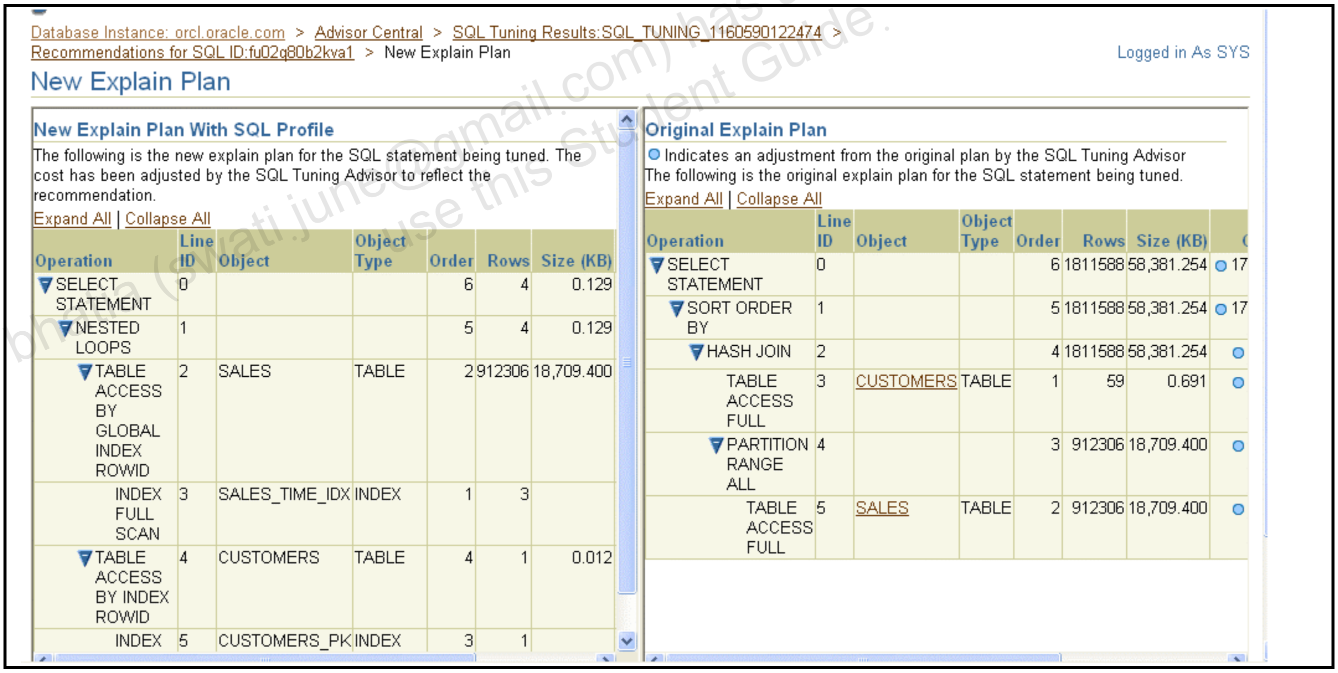
SQL Text

select time_id, QUANTITY_SOLD, AMOUNT_SOLD from sales s, customers c where c.cust_id = s.cust_id and CUST_FIRST_NAME='Dina' order by time_id

Select Recommendation

| Select | Type | Findings | Recommendations | Rational | Benefit (%) | New Explain Plan |
|---|---|---|---|---|---|---|
| ◉ | SQL Profile | A potentially better execution plan was found for this statement. | Consider accepting the recommended SQL profile. | | 99.95 | 👓 |

5. As you can see, the new explain plan removes the full table scans. Click the **Back** button in the browser to return to the previous page.



Database Instance: orcl.oracle.com > Advisor Central > SQL Tuning Results:SQL_TUNING_1160590122474 > Recommendations for SQL ID:fu02q80b2kva1 > New Explain Plan

Logged in As SYS

New Explain Plan

New Explain Plan With SQL Profile

The following is the new explain plan for the SQL statement being tuned. The cost has been adjusted by the SQL Tuning Advisor to reflect the recommendation.

Expand All | Collapse All

| Operation | Line ID | Object | Object Type | Order | Rows | Size (KB) |
|---|---|---|---|---|---|---|
| ▼SELECT STATEMENT | 0 | | | 6 | 4 | 0.129 |
| ▼NESTED LOOPS | 1 | | | 5 | 4 | 0.129 |
| ▼TABLE ACCESS BY GLOBAL INDEX ROWID | 2 | SALES | TABLE | 2 | 912306 | 18,709.400 |
| INDEX FULL SCAN | 3 | SALES_TIME_IDX | INDEX | 1 | 3 | |
| ▼TABLE ACCESS BY INDEX ROWID | 4 | CUSTOMERS | TABLE | 4 | 1 | 0.012 |
| INDEX | 5 | CUSTOMERS_PK | INDEX | 3 | 1 | |

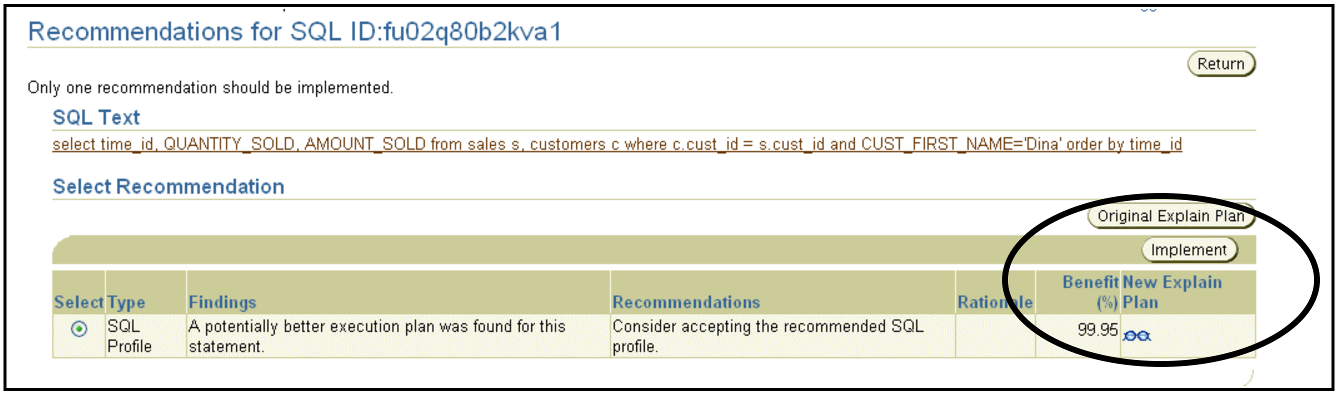
Original Explain Plan

○ Indicates an adjustment from the original plan by the SQL Tuning Advisor
The following is the original explain plan for the SQL statement being tuned.

Expand All | Collapse All

| Operation | Line ID | Object | Object Type | Order | Rows | Size (KB) | |
|---|---|---|---|---|---|---|---|
| ▼SELECT STATEMENT | 0 | | | 6 | 1811588 | 58,381.254 | ○ 17 |
| ▼SORT ORDER BY | 1 | | | 5 | 1811588 | 58,381.254 | ○ 17 |
| ▼HASH JOIN | 2 | | | 4 | 1811588 | 58,381.254 | ○ |
| TABLE ACCESS FULL | 3 | CUSTOMERS | TABLE | 1 | 59 | 0.691 | ○ |
| ▼PARTITION RANGE ALL | 4 | | | 3 | 912306 | 18,709.400 | ○ |
| TABLE ACCESS FULL | 5 | SALES | TABLE | 2 | 912306 | 18,709.400 | ○ |

6. Click the **Implement** button to implement the tuning recommendation.



Recommendations for SQL ID:fu02q80b2kva1

Only one recommendation should be implemented.

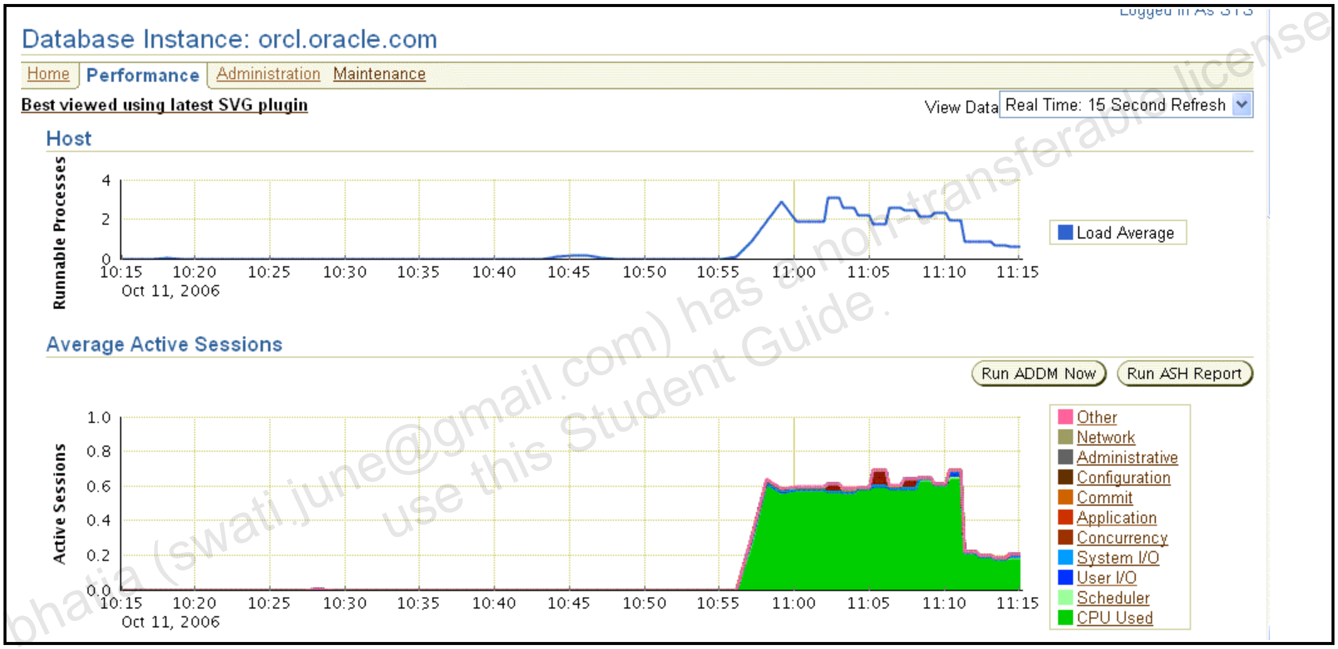
SQL Text

select time_id, QUANTITY_SOLD, AMOUNT_SOLD from sales s, customers c where c.cust_id = s.cust_id and CUST_FIRST_NAME='Dina' order by time_id

Select Recommendation

| Select | Type | Findings | Recommendations | Rationale | Benefit (%) | New Explain Plan |
|---|---|---|---|---|---|---|
| ● | SQL Profile | A potentially better execution plan was found for this statement. | Consider accepting the recommended SQL profile. |  | 99.95 | 👓 |

7. A confirmation page appears indicating that the SQL Profile was successfully created. Click the **Database** breadcrumb and then the Performance tab.



Database Instance: orcl.oracle.com

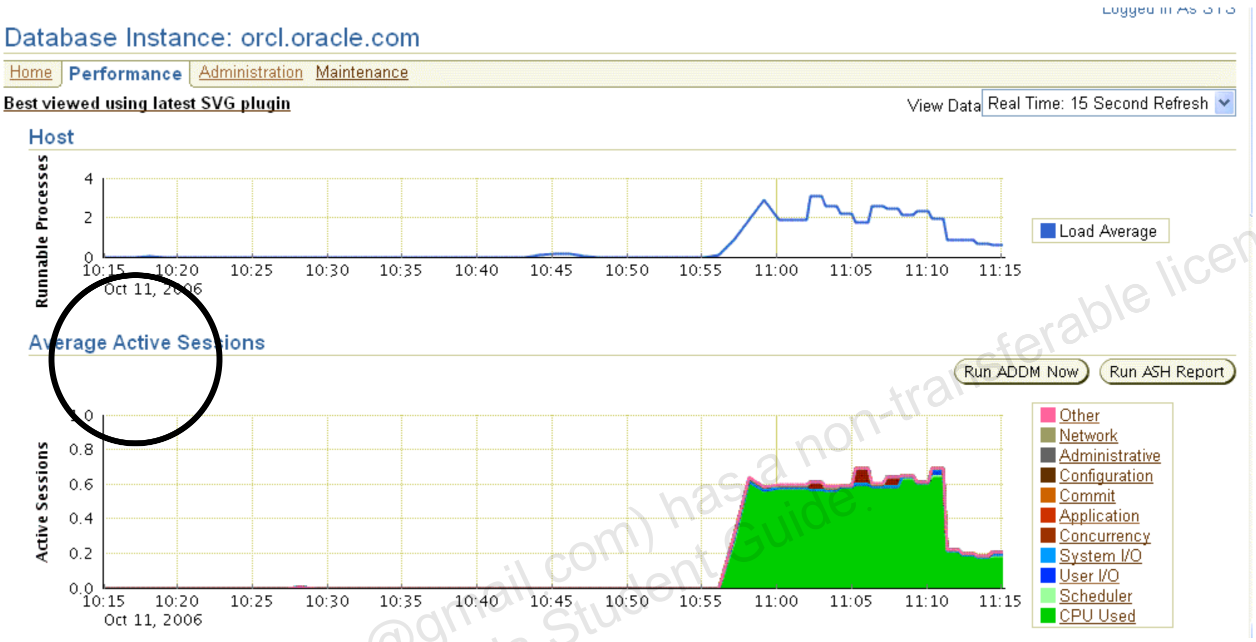
Best viewed using latest SVG plugin

View Data Real Time: 15 Second Refresh

### Reviewing SQL Execution Details for a SQL Statement

Now that you have implemented the tuning suggestion, review the SQL statement and its execution details. Perform the following steps.

1. Click the **Performance** tab.
2. Scroll down to the **Average Active Sessions** chart. Wait for about **one minute** and observe how the User I/O is decreasing.



3. To clean up your environment, execute the following command from your command-line window.

```
$ ./cleanup_perflab.sh
```

4. Close your browser.

## Workshop 3: Access Paths

### Objectives

- Tune SQL statements by using the AND operator
- Tune SQL statements using the OR and IN operators
- Set up and understand concatenated indexes
- Learn the benefits from bitmapped indexes
- Use function-based indexes
- Use join indexes
- Use bitmap join indexes
- Use index monitoring
- Use fast full index scans
- Use full scan of indexes

### Introduction

In this workshop, the SQL statements have a compound WHERE clause, consisting of multiple predicates combined with AND and OR operators. You can tune these statements by creating indexes. You also investigate how the optimizer can benefit from creating concatenated indexes, bitmapped indexes, and function-based indexes.

This workshop can be done entirely in the SQL*Plus environment. Try to work in small groups and discuss the workshop results. Each time you load a new SQL statement, try to predict what the optimizer will do before running the statement. Take notes during the workshop as an aid for the wrap-up discussion.

To obtain an explain plan without executing the query for any of the queries provided in the scripts for this workshop, you can use the following steps:

1. Open a terminal window. Change directories to the workshop directory. Log in to SQL*Plus as SH with the password of SH. Drop all existing indexes on the CUSTOMERS table and then create indexes on the following columns: CUST_GENDER, CUST_POSTAL_CODE, and CUST_CREDIT_LIMIT using the **ci.sql** script. Provide table names and column names when prompted. Indexes are automatically named for you. Check the indexes on the CUSTOMERS table using the **li.sql** script.

```
$cd workshop
$sqlplus sh/sh
SQL> @dai
on which table: customers
SQL> @ci
on which table : customers
on which column(s): cust_gender
Creating index cust_cust_gender_idx …
SQL> @ci
on which table : customers
on which column(s): cust_postal_code
```

```
Creating index cust_cust_postal_code_idx …
SQL> @ci
on which table : customers
on which column(s): cust_credit_limit
Creating index cust_cust_credit_limit_idx …
SQL> @li
List indexes on table: customers
TABLE_NAME    INDEX_TYPE           INDEX_NAME
------------- -------------------- -------------------------
CUSTOMERS     UNIQUE               CUSTOMERS_PK
              NONUNIQUE            CUST_CUST_CREDIT_LIMIT_IDX
                                   CUST_CUST_GENDER_IDX
                                   CUST_CUST_POSTAL_CODE_IDX
```

Run **im.sql** to start monitoring these indexes:

```
SQL> get im
  1  ALTER INDEX &indexname MONITORING USAGE;
SQL> @im
Enter value for indexname:         CUSTOMERS_PK
Index altered.
SQL> @im
Enter value for indexname:         CUST_CUST_CREDIT_LIMIT_IDX
Index altered.
SQL> @im
Enter value for indexname:         CUST_CUST_GENDER_IDX
Index altered.
SQL> @im
Enter value for indexname:         CUST_CUST_POSTAL_CODE_IDX
Index altered.
```

**Student Observations:**

2. Enable AUTOTRACE and get query **ws_03_01.sql**. The WHERE clause contains three predicates. Execute this statement and take notes about the indexes used, the cost of the execution plan, and the amount of I/O performed.

```
SQL> @atto
SQL> get ws_03_01
1 SELECT c.*
2 FROM customers c
3 WHERE cust_gender = 'M'
4 AND cust_postal_code = 40804
5* AND cust_credit_limit = 10000
SQL> @ws_03_01
```

**Student Observations:**

3. Disable AUTOTRACE. Disable index monitoring by running **nm.sql**. Run **ws_03_03b.sql** to see which indexes are used by retrieving from V$OBJECT_USAGE.

```
SQL>@atoff
SQL> get nm.sql
  1 ALTER INDEX &indexname NOMONITORING USAGE;
SQL>@nm
Enter value for indexname:         CUST_CUST_CREDIT_LIMIT_IDX
Index altered.
SQL> /
Enter value for indexname:         CUST_CUST_GENDER_IDX
Index altered.
SQL> /
Enter value for indexname:         CUST_CUST_POSTAL_CODE_IDX
Index altered.
SQL>  get ws_03_03b
  1 SELECT * FROM v$object_usage
SQL>  @ws_03_03b
```

4. Drop the indexes again and replace them with a single concatenated index, then run **ws_03_01.sql** again. Also change the AUTOTRACE setting to suppress statement results. Enable index monitoring again.

```
SQL> @dai
on which table: customers
SQL> @ci
on which table : customers
on which column(s): cust_gender, cust_credit_limit, cust_postal_code
SQL>@im
Enter value for indexname:  CUST_CUST_GENDER_CUST_CRE_IDX

SQL> @atto
SQL> get ws_03_01
1 SELECT c.*
2 FROM customers c
3 WHERE cust_gender = 'M'
4 AND cust_postal_code = 40804
5* AND cust_credit_limit = 10000
SQL> @ws_03_01
```

**Student Observations:**

5. Disable AUTOTRACE. Disable index monitoring by running nm.sql. Run **ws_03_03b.sql** to see which indexes are used by retrieving from **V$OBJECT_USAGE**.

```
SQL>@atoff
SQL> get nm.sql
  1 ALTER INDEX &indexname NOMONITORING USAGE;
SQL> @nm
Enter value for indexname: customers_pk
Index altered.
SQL> /
Enter value for indexname: CUST_CUST_GENDER_CUST_CRE_IDX
Index altered.
```

```
SQL>@atoff
SQL>  get ws_03_03b
  1 SELECT * FROM v$object_usage
SQL>  @ws_03_03b
```

6. Enable AUTOTRACE again. Now investigate what happens when not all columns of a concatenated index are present in a predicate. Get **ws_03_02.sql** and compare it with the original statement in **ws_03_01.sql**.

```
SQL>@atto
SQL> get ws_03_02
 1 select
 2 c.*
 3 FROM customers c
 4 WHERE cust_gender = 'M'
 5* AND cust_credit_limit = 10000
SQL> /
```

**Student Observations:**

The index usage depends on selectivity of the column, which is very poor for the CUST_CREDIT_LIMIT column.

Get **ws_03_03.sql** and compare it with the original statement in **ws_03_01.sql**. Now the predicate on the CUST_CREDIT_LIMIT column is removed.

```
SQL> get ws_03_03
 1 select
 2 c.*
 3 FROM customers c
 4 WHERE cust_gender = 'M'
 5* AND cust_postal_code = 40804
SQL> /
```

Get **ws_03_04.sql** and compare it with the original statement in **ws_03_01.sql**. Now the predicate on the CUST_GENDER column is removed.

```
SQL> get ws_03_04
 1 SELECT
 2 c.*
 3 FROM customers c
 4 WHERE cust_postal_code = 40804
 5* AND cust_credit_limit = 10000
 SQL> /
```

**Student Observations:**

7. You should investigate one more option: Drop all indexes on the CUSTOMERS table and create three bitmapped indexes using the **cbinp.sql** script. Start **ws_03_01.sql** again and edit the statement to specify an INDEX_COMBINE (**ws_03_01b.sql** but do not

provide an index name) hint when needed to force bitmapped index usage. Compare the results with the concatenated index approach.

```
SQL> @dai
on which table: customers
SQL> @cbinp
on which table : customers
on which column(s): cust_gender
SQL> @cbinp
on which table : customers
on which column(s): cust_postal_code
SQL> @cbinp
on which table : customers
on which column(s): cust_credit_limit
SQL>@li
List indexes on table :  Customers
SQL>@atto
SQL> @ws_03_01
```

**Student Observations:**

```
SQL>@ws_03_01b
Enter value for index_name:
```

**Student Observations:**

The optimizer may not use the bitmap indexes. You may provide hints to the optimizer to use the index. Use the **`ws_03_01b.sql`** script to provide the different indexes as hints and take notes on the behavior.

```
SQL>@ws_03_01b
Enter value for index_name:  CUST_CUST_CREDIT_LIMIT_IDX

SQL>@ws_03_01b
Enter value for index_name:  CUST_CUST_GENDER_IDX

SQL>@ws_03_01b
Enter value for index_name:  CUST_CUST_POSTAL_CODE_IDX
```

**Student Observations:**

8. Drop all indexes on the `CUSTOMERS` table. Make sure that you have a normal B*-tree index on the `CUST_YEAR_OF_BIRTH` and `CUST_CREDIT_LIMIT` columns. Now get **`ws_03_05.sql`**. This time you see two predicates combined with an `OR` operator.

```
SQL> @dai
on which table: customers
SQL> @ci
on which table : customers
on which column(s): cust_year_of_birth
SQL> @ci
on which table : customers
on which column(s): cust_credit_limit
SQL> get ws_03_05
1 SELECT c.*
2 FROM customers c
3 WHERE c.cust_year_of_birth = 1953
4* AND c.cust_credit_limit = 10000
SQL> @ws_03_05
```

**Student Observations:**

You see that both indexes are used and combined with a BITMAP AND operator. Investigate what happens if you drop the index on the CUST_YEAR_OF_BIRTH column:

```
SQL> @atoff
SQL> drop index cust_cust_year_of_birth_idx;
SQL> @atto
SQL> @ws_03_05
```

This time the optimizer apparently prefers to perform a full table scan. Why?

**Student Observations:**

9. Drop all indexes on the CUSTOMERS table. Create bitmapped indexes on CUST_YEAR_OF_BIRTH, CUST_POSTAL_CODE, and CUST_CREDIT_LIMIT, and then get the query **ws_03_07.sql**. This statement has a complicated WHERE clause. Bitmapped indexes are good for this type of statement; you see several bitmap operations in the execution plan.

```
SQL> @dai
on which table: customers
SQL> @cbinp
on which table : customers
on which column(s): cust_year_of_birth
SQL> @cbinp
on which table : customers
on which column(s): cust_postal_code
SQL> @cbinp
on which table : customers
on which column(s): cust_credit_limit
SQL> @atto
SQL> get ws_03_07
1 SELECT c.*
2 FROM customers c
3 WHERE (c.cust_year_of_birth = '1970' AND
4 c.cust_postal_code = 40804)
5* AND NOT cust_credit_limit = 15000
SQL>@ws_03_07
```

**Student Observations:**

10. Get and run query **`ws_03_08.sql`**.

```
SQL>@atto
SQL> get ws_03_08
1 SELECT c.*
2 FROM customers c
3* WHERE cust_id in (88340,104590,44910)
SQL> @ws_03_08
```

The optimizer uses the primary key index. This is possible because `CUST_ID` is the leading column of this index.

**Student Observations:**

11. Drop all indexes and create a normal B*-tree index on the `COUNTRY_ID` column. Start query **`ws_03_09.sql`**. Make a note about the estimated cost. The `COUNTRY_ID` column in the `CUSTOMERS` table contains skewed data (**`ws_03_10.sql`** displays how skewed the data is). Verify that a histogram has been created for the `COUNTRY_ID` column (use `colhist.sql`) and run **`ws_03_09.sql`** again. Note the estimated cost.

```
SQL> @dai
on which table: customers
SQL> @ci
on which table : customers
on which column(s): country_id
SQL>@atto
SQL> get ws_03_09
  1  SELECT s.*
  2  FROM customers s
  3* WHERE country_id in (52790, 52771)
SQL> @ws_03_09
SQL> @atoff
SQL> get ws_03_10
  1  SELECT country_id, count(*)
  2 FROM customers
  3 GROUP BY country_id
SQL> @ws_03_10
SQL> @colhist
SQL> @atto
SQL> @ws_03_09
```

Note what happens when you change `52790` to `52791`.

```
SQL> l
SQL> 3
SQL> c/52790/52791
SQL>/
```

Note the cost listed with the execution plan.

The optimizer can make this intelligent decision based on the histogram statistics.

**Student Observations:**

12. Next, investigate the benefits of fast full index scans. Drop all indexes on the CUSTOMERS table and create a new concatenated index on the CUST_LAST_NAME and CUST_FIRST_NAME columns.

```
SQL> @dai
on which table: customers
SQL> @atto
SQL> get ws_03_11
 1 SELECT c.cust_last_name
 2 , c.cust_first_name
 3* FROM customers c
SQL> @ws_03_11
SQL> @ci
on which table : customers
on which column(s): cust_last_name, cust_first_name

SQL> @ws_03_11
```

As you see, the optimizer benefits from a full index scan. Remember that the Oracle Server uses multiblock reads but does not guarantee any ordering.

**Student Observations:**

Run the **ws_03_11a.sql** query. Do you notice any difference in plans?

```
SQL> get ws_03_11a
  1  SELECT cust_first_name
  2  FROM   customers c
  3* WHERE cust_first_name = 'Victoria'
SQL> @ws_03_11a
```

**Student Observations:**

Drop all indexes and create two separate B*-tree indexes on the last and first name columns. Run **ws_03_11.sql** again and view the execution plan.

```
SQL> @dai
on which table: customers
SQL> @ci
on which table : customers
on which column(s): cust_last_name
SQL> @ci
on which table : customers
on which column(s): cust_first_name
SQL> @ws_03_11
```

**Student Observations:**

Try running the same query with an `INDEX_JOIN` hint and see what happens. You can use **`ws_03_11b.sql`** to do this.

```
SQL>get ws_03_11b
  1  SELECT /*+index_join(c  cust_cust_first_name_idx
       cust_cust_last_name_idx)/
  2  c.cust_last_name
  3  ,      c.cust_first_name
  4* FROM   customers c
SQL> @ws_03_11b
```

**Student Observations:**

13. Now examine the `COUNT` function. The `COUNT` function can benefit from bitmapped indexes by counting the number of ones in a bitmap, which is an efficient operation. Get and run **`ws_03_13.sql`**. Then create a bitmap index on `CUST_CREDIT_LIMIT` and try again.

```
SQL>@dai
on which table : customers
SQL> @atto
SQL> get ws_03_13
  1 SELECT count(*) credit_limit
  2 FROM customers
  3* WHERE cust_credit_limit = 10000;
SQL> @ws_03_13
SQL> @cbinp
on which table : customers
on which column: cust_credit_limit
SQL> @ws_03_13
```

**Student Observations:**

If you have some time left, replace the bitmapped index with a normal one. Then you see that a normal index also improves performance; however, the bitmapped index is more efficient.

14. Finally, investigate the benefits of function-based indexes. Drop all indexes on the CUSTOMERS table. Create an index on the CUST_LAST_NAME column and start the query **ws_03_14.sql**.

```
SQL> @dai
on which table: customers
SQL>@ci

on which table : customers
on which column: cust_last_name
SQL> @atto
SQL> get ws_03_14
1 SELECT cust_id, country_id
2 FROM customers
3* WHERE lower( cust_last_name) like 'gentle'
SQL> @ws_03_14
```

**Student Observations:**

Create a function-based index (**ws_03_14a.sql**) that utilizes the LOWER function on the CUST_LAST_NAME column. Rerun **ws_03_14.sql** and compare the results.

```
SQL> create index lower_cust_last_name_idx on
  2> customers(lower(cust_last_name));
SQL> @ws_03_14
```

**Student Observations:**

## Workshop Summary

After completing this workshop, you should have learned the following:

- Up to five indexes can be merged to optimize predicates combined with AND.
- Concatenated indexes usually offer better performance because they are premerged. You do not need predicates on each column of the concatenated index; however, make sure that the leading column is specified.
- Be careful with OR constructs: As soon as the left or right side is unindexed, the index on the other side is useless.
- Bitmapped indexes are efficient in case of complex WHERE clauses with many AND, NOT, and OR constructs. This is based on fast internal bit-level operations.
- The COUNT function can benefit from bitmapped indexes by using an efficient built-in operator that counts the number of ones in a bitmap.
- Histograms enable the optimizer to estimate execution plan costs better, particularly when the data is skewed.
- Fast full index scans are an alternative to full table scans when an index contains all the columns that are needed for a query. Fast full index scans cannot be used to eliminate a sort operation. They read the entire index by using multiblock I/O.
- Function-based indexes can facilitate processing queries.

### Workshop 4: Using Index-Organized Tables

### Workshop Objectives

- Examine the properties of an index-organized table
- Measure performance differences using index-organized tables

This workshop can be done entirely in the SQL*Plus environment. Try to work in small groups and discuss the workshop results. Each time you load a new SQL statement, try to predict what the optimizer will do before running the statement. Take notes during the workshop as an aid for the wrap-up discussion.

1. Open a terminal window. Change to workshop directory. Start SQL*Plus. Log in as **SH/SH**.
2. Run the **ws_04_02.sql** script to create an index-organized table on the PROMO_ID column in the PROMOTIONS table, and populate it with data from the PROMOTIONS table.

```
$cd workshop
$sqlplus sh/sh

SQL> get ws_04_02
  1  CREATE table promotions_iot
  2  (promo_id number primary key
  3  ,  promo_name VARCHAR2(40)
  4  ,  promo_subcategory VARCHAR2 (30)
  5  ,  promo_category  VARCHAR2 (30)
  6  ,  promo_cost NUMBER
  7  ,  promo_begin_date DATE
  8  ,  promo_end_date DATE)
  9  ORGANIZATION INDEX
 10  /
 11  INSERT INTO promotions_iot
 12  SELECT promo_id, promo_name, promo_subcategory, promo_category,
 13  promo_cost   promo_begin_date, promo_end_date
 14  FROM promotions
 15* /
 SQL> @ws_04_02
```

Analyze the execution plan for query on the PROMOTIONS table in the **ws_04_03.sql** script and compare it to the same query on the PROMOTIONS_IOT table in the **ws_04_04.sql** script.

```
SQL> @attox
SQL> get ws_04_03
  1  SELECT *
  2  FROM promotions
  3* WHERE promo_id > 300
SQL> @ws_04_03
SQL> get ws_04_04
  1  SELECT *
  2  FROM promotions_iot
  3* WHERE promo_id > 300
SQL> @ws_04_04
```

**Student Observations:**

3. Create an index on the PROMO_SUBCATEGORY column in the PROMOTIONS_IOT table, and then run **ws_04_05.sql** and compare the results.

```
SQL> @ci
on which table : promotions_iot
on which column(s): promo_subcategory
SQL> get ws_04_05
  1  SELECT *
  2  FROM promotions_iot
  3* WHERE promo_subcategory = 'online discount'
 SQL>@ws_04_05
```

**Student Observations:**

4. Turn AUTOTRACE off. Now take a look at dynamic sampling. First create a table using the script **ws_04_06.sql**:

```
SQL> @atoff
SQL> get ws_04_06.sql
  1  CREATE TABLE sales_temp
  2  (cust_id number,
  3   amount_sold number)
  4   /
  6  Insert into sales_temp SELECT cust_id, amount_sold
  7   FROM sales WHERE rownum <= 10
SQL> @ws_04_06

Table created
10 rows created.
```

Get and run **ws_04_07.sql**. This creates an EXPLAIN PLAN for a query that selects from the SALES_TEMP table and DUAL table. View the explain plan using **rp.sql**.

```
SQL>@atoff
SQL> get ws_04_07
  1  explain plan for
  2* SELECT * FROM sales_temp, dual
SQL>@ws_04_07
SQL> @rp
```

**Student Observations:**

As you can see, the optimizer uses dynamic sampling to create the execution plan for the query. Try gathering statistics on the SALES_TEMP table. You can use **ws_04_08.sql** for this. Run **ws_04_07.sql** and **rp.sql** again. Do you see any difference?

```
SQL> get ws_04_08
  1* exec dbms_stats.gather_table_stats
  2 ('SH','SALES_TEMP');
SQL> @ws_04_08
SQL> get ws_04_07
  1  explain plan for
  2* SELECT * FROM sales_temp, dual
SQL> @ws_04_07
SQL> @rp
```

**Student Observations:**

Add 100,000 rows to the SALES_TEMP table by using **ws_04_09.sql**. Run **ws_04_07.sql** and **rp.sql** again:

```
SQL> get ws_04_09
  1  Insert into sales_temp SELECT cust_id, amount_sold
  2* FROM sales WHERE rownum <= 100000;
SQL>  @ws_04_09
SQL> get ws_04_07
  1  explain plan for
  2*   SELECT * FROM sales_temp, dual
SQL> @ws_04_07
SQL> @rp
```

**Student Observations:**

The optimizer still works on the old statistics. Run **ws_04_11.sql** to delete the statistics:

```
SQL> get ws_04_11
  1* execute dbms_stats.delete_table_stats('SH','SALES_TEMP');
  SQL> @ws_04_11
  SQL> get ws_04_07
    1  explain plan for
    2* SELECT * FROM sales_temp, dual
  SQL> @ws_04_07
  SQL> @rp
```

Does the optimizer use dynamic sampling again?

### Workshop Summary

After completing this workshop, you should have learned the following:

- Because data rows are stored in the index, index-organized tables provide faster key-based access to table data for queries that involve exact match or range search (or both).
- The Oracle Server constructs secondary indexes on index-organized tables using logical row identifiers (logical rowids) that are based on the table's primary key.
- This logical rowid optionally includes a physical guess, which identifies the block location of the row.

The Oracle Server can use these guesses to probe directly into the leaf block of the index-organized table, bypassing the primary key search. Be aware that because rows in index-organized tables do not have permanent physical addresses, the guesses can become old when rows are moved to new blocks.

## Workshop 5: Materialized Views

## Part A: Examining Query Rewrite and Materialized View Capabilities

### Workshop Objectives

- Examining the performance of views
- Examining the difference between mergeable and nonmergeable views
- Examining the properties of materialized views
- Examining the performance of materialized views

### Introduction

In this workshop, you focus on queries based on views and their impact on query performance. You examine the characteristics of mergeable views and nonmergeable views, and you create a materialized view and analyze its performance.

This workshop can be done entirely in the SQL*Plus environment. Try to work in small groups and discuss the workshop results. Each time you load a new SQL statement, try to predict what the optimizer will do before running the statement. Take notes during the workshop as an aid for the wrap-up discussion.

To obtain an explain plan without executing the query for any of the queries provided in the scripts for this workshop, you can use the following steps:

1. Open a terminal window. Change to workshop directory. Start SQL*Plus. Log in as **`SH/SH`**. Run the SQL statement in **`ws_05_01.sql`**. This script creates a view called `V_SALES_DETAIL`, based on an outer join.

```
$cd workshop
$sqlplus sh/sh

SQL> get ws_05_01
  1 create or replace view v_sales_detail
  2 as
  3 SELECT sa.prod_id
  4 , pr.prod_name
  5 , pr.supplier_id
  6 , pr.prod_status
  7 , sa.amount_sold
  8 , to_char(sa.time_id, 'YYYY-MON-DD') date_sold
  9 FROM sales sa
  10 , products pr
  11* WHERE pr.prod_id = sa.prod_id(+)
SQL> @ws_05_01
SQL> describe v_sales_detail
```

**Student Observations:**

2. Drop indexes on the SALES and PRODUCTS tables. Compare the SQL statements in **ws_05_02.sql** and **ws_05_03.sql**.

```
SQL> @dai
on which table: sales
SQL> @dai
on which table: products
SQL> set timing on
SQL> @atto
SQL> get ws_05_02
  1  SELECT v.prod_name
  2  , v.prod_status
  3  , v.date_sold
  4  FROM v_sales_detail v
  5*  WHERE v.prod_id = 120
SQL> @ws_05_02
SQL> get ws_05_03
  1  SELECT pr.prod_name
  2  , pr.prod_status
  3  , to_char(sa.time_id, 'YYYY-MON-DD') date_sold
  4  FROM sales sa
  5  , products pr
  6  WHERE pr.prod_id = sa.prod_id(+)
  7* AND sa.prod_id = 120
SQL> @ws_05_03
```

These two statements produce the same rows (although they may be sorted differently). Is there any difference in their execution plans?

**Student Observations:**

3. Now compare the SQL statements in **ws_05_04.sql** and **ws_05_05.sql**.

**Note:** This may take some time.

```
SQL> get ws_05_04
  1  SELECT prod_name
  2  ,      prod_status
  3  ,       date_sold
  4  FROM   v_sales_detail
  5* WHERE  prod_id = 120
SQL> @ws_05_04

SQL> get ws_05_05
 1  SELECT prod_name
  2  ,      prod_status
  3  ,      date_sold
  4  FROM   v_sales_detail
  5* WHERE  prod_id  in (50, 120)
SQL> @ws_05_05
```

Analyze these two statements. Only the WHERE clause has changed. Why are the execution plans different?

**Student Observations:**

4. Compare the SQL statements in **ws_05_02.sql** and **ws_05_06.sql**:

```
SQL> get ws_05_02
 1 SELECT v.prod_name
 2 , v.prod_status
 3 , v.date_sold
 4 FROM v_sales_detail v
 5* WHERE v.prod_id = 120
SQL> @ws_05_02
SQL> get ws_05_06
 1 SELECT v.prod_name
 2 , v.prod_status
 3 , v.date_sold
 4 FROM v_sales_detail v
 5* WHERE v.prod_name = 'Sunburst Dress'
SQL> @ws_05_06
```

Why are the execution plans different?

**Student Observations:**

5. Drop all indexes on the CUSTOMERS table. Run the SQL statement in **ws_05_07.sql**. This script creates a view called CUST_INC_VIEW based on customers with a CUST_INCOME_LEVEL like 'B%'.

```
SQL> @dai
On which table: customers
SQL> get ws_05_07
  1  create or replace view cust_inc_view as
  2  SELECT cust_id, cust_credit_limit,
  3  cust_last_name, cust_income_level
  4  FROM customers
  5* WHERE cust_income_level like 'B%'
 SQL> @ws_05_07
```

Consider the query in **ws_05_08.sql** that accesses the view. The query selects the customers with a credit limit of 1,500 or the customer with the ID of 499290 from the list of customers in the B income level.

```
SQL> get ws_05_08
  1  SELECT  cust_last_name FROM cust_inc_view
  2  WHERE cust_credit_limit = 1500
  3* or cust_id =  499290
SQL> @ws_05_08
```

**Student Observations:**

Create indexes on CUST_CREDIT_LIMIT and CUST_INCOME_LEVEL and view the execution plan again:

```
SQL> @ci
     on which table    : customers
     on which column(s): cust_credit_limit
SQL> @ci
     on which table    : customers
     on which column(s): cust_income_level
```

**Student Observations:**

6. Run the SQL statement in **ws_05_09.sql**. This script creates a view called V_AVG_CREDIT_LIMIT based on the average credit limits per country.

```
SQL> get ws_05_09
  1  create or replace view v_avg_credit_limit
  2  as SELECT country_id
  3  , avg(cust_credit_limit) AVG_CREDIT
  4  FROM customers
  5* GROUP BY country_id
SQL> @ws_05_09
```

Consider the query in **ws_05_10.sql**, which accesses the view. The query selects the average credit limits for countries in a specific country or region.

```
SQL> get ws_05_10
 1   SELECT country_name   avg_credit
 2   FROM v_avg_credit_limit a, countries c
 3   WHERE a.country_id = c.country_id
 4*  AND country_region = 'Europe'
SQL> @ws_05_10
```

**Student Observations:**

View the execution plan . Notice how the optimizer joins the CUSTOMERS and COUNTRIES tables and creates a GROUP BY operation. Create an index on the COUNTRY_ID column in the CUSTOMERS table, and then rerun the **ws_05_10.sql** script. Is the index used?

```
SQL> @dai
on which table: customers
SQL> @ci
on which table : customers
on which column(s): country_id
SQL> @ws_05_10
```

**Student Observations:**

7. Run the SQL statement in **`ws_05_11.sql`**. This script creates a view that contains a `SELECT` statement with the `UNION` operator. Run the **`ws_05_12.sql`** script to query data from the view.

```
 SQL> get ws_05_11
  1  create or replace view v_99_00_times
  2  as
  3  SELECT *
  4  FROM times
  5  WHERE calendar_year = '1999'
  6  union
  7  SELECT *
  8  FROM times
  9* WHERE calendar_year = '2000'
SQL> @ws_05_11
SQL> get ws_05_12
  1  SELECT calendar_month_number
  2  , calendar_month_name
  3  , calendar_quarter_desc
  4  , calendar_quarter_number
  5  FROM v_99_00_times
  6* WHERE calendar_year = '1999'
SQL> @ws_05_12
```

**Student Observations:**

Create an index on the `CALENDAR_YEAR` column in the `TIMES` table. Rerun the **`ws_05_12.sql`** script and compare the results. Is the index used? Why or why not?

```
SQL> @dai
on which table: times
SQL> @ci
on which table : times
on which column(s): calendar_year

SQL> get ws_05_12
  1  SELECT calendar_month_number
  2  , calendar_month_name
  3  , calendar_quarter_desc
  4  , calendar_quarter_number
  5  FROM v_99_00_times
  6*  WHERE calendar_year = '1999'
SQL> @ws_05_12
```

**Student Observations:**

8. Run the SQL statement in **`ws_05_13.sql`.** This script creates a materialized view called `JOIN_SALES_TIME_PRODUCT_MV`.

```
SQL> get ws_05_13
  1  create materialized view join_sales_time_product_mv
  2  enable query rewrite
  3  as
  4  SELECT p.prod_id, p.prod_name, t.time_id,
  5         s.channel_id, s.promo_id, s.cust_id,
  6         t.week_ending_day, s.amount_sold
  7  FROM   sales s, products p, times t
  8  WHERE  s.time_id=t.time_id
  9* AND    s.prod_id = p.prod_id
sql> @ws_05_13
```

**Student Observations:**

Consider the query in **`ws_05_14.sql`**, which accesses the materialized view. The query selects data from the materialized view for a specific year.

```
SQL> get ws_05_14
  1  SELECT *
  2  FROM join_sales_time_product_mv
  3* WHERE amount_sold = 2000
SQL> @ws_05_14
```

**Student Observations:**

Create an index on the `AMOUNT_SOLD` column in the materialized view, then rerun the **`ws_05_14.sql`** script and compare the results.

**Note:** This may take some time.

```
SQL> @ci
on which table : join_sales_time_product_mv
on which column(s): amount _sold
SQL> @ws_05_14
```

**Student Observations:**

9. Get and run the **ws_05_17.sql** query to create the COSTS_PER_VIEW_MV materialized view:

```
SQL> get ws_05_17.sql
  1  CREATE MATERIALIZED VIEW
  2  costs_per_year_mv
  3  ENABLE QUERY REWRITE
  4  AS
  5  SELECT   t.week_ending_day
  6  ,        t.calendar_year
  7  ,        p.prod_subcategory
  8  ,        sum(c.unit_cost) AS dollars
  9  FROM     costs c
 10  ,        times t
 11  ,        products p
 12  WHERE    c.time_id = t.time_id
 13  AND      c.prod_id = p.prod_id
 14  GROUP BY t.week_ending_day
 15  ,        t.calendar_year
 16* ,        p.prod_subcategory
 17 CREATE INDEX mvi_dollars on costs_per_year_mv(dollars);
 18
 19 CREATE INDEX mvi_calendar_year  ON
 20 costs_per_year_mv(calendar_year);
 21
 22 Exec DBMS_STATS.GATHER_TABLE_STATS -
 23 ('sh','costs_per_year_mv');

 SQL>@ws_05_17.sql
```

Enable AUTOTRACE. Now view and execute the **ws_05_18.sql** script. What do you see? View and execute **ws_05_18b.sql**. Compare with the previous results.

```
SQL> @atto
SQL> get ws_05_18
  1  set autotrace traceonly explain
  2  SELECT   t.week_ending_day
  3  ,        t.calendar_year
  4  ,        p.prod_subcategory
  5  ,        sum(c.unit_cost) AS dollars
  6  FROM     costs c, times t, products p
  7  WHERE    c.time_id = t.time_id
  8  AND      c.prod_id = p.prod_id
  9  AND      t.calendar_year = '1999'
 10  GROUP BY t.week_ending_day, t.calendar_year
 11  ,        p.prod_subcategory
 12  HAVING   sum(c.unit_cost) > 10000;
SQL> @ws_05_18
```

**Student observations:**

```
SQL> get ws_05_18b
  1  SELECT   week_ending_day
  2  ,        prod_subcategory
  3  ,        dollars
  4  FROM     cost_per_year_mv
  5  WHERE    calendar_year = '1999'
  6* AND      dollars > 10000;
SQL> @ws_05_18b
```

**Student observations:**

Now verify the properties of the COSTS_PER_YEAR_MV by creating entries in the MV_CAPABILITIES_TABLE. You can use the scripts **ws_05_19.sql** and **ws_05_20.sql** to do this.

**Note:** If this table has not been created, you can create it by running the **utlxmv.sql** script.

```
SQL>@ utlxmv.sql
SQL> EXEC dbms_mview.explain_mview ('costs_per_year_mv', '678');
PL/SQL procedure successfully completed.
SQL> get ws_05_19
  1* EXEC DBMS_MVIEW.EXPLAIN_MVIEW ('costs_per_year_mv', '678');
SQL> @ws_05_19
SQL>get ws_05_20
  1  SELECT mvname, capability_name, possible
  2  FROM mv_capabilities_table
  3* WHERE statement_id = '678' ORDER BY seq
SQL>@ws_05_20
```

10. View and execute the **ws_05_21.sql** script to create a materialized view called MY_MV, and execute the dbms_stats.gather_table_stats (USER, 'MY_MV') procedure to gather statistics against MY_MV.

```
SQL>Get ws_05_21
  1   create materialized view my_mv
  2     build immediate
  3     enable query rewrite
  4  as SELECT prod_id
  5     ,      avg(amount_sold) as avg_amount
  6     FROM   sales
  7     WHERE  channel_id = 9
  8     group  by prod_id;
SQL>@ws_05_21
Materialized view created.

SQL> execute dbms_stats.gather_table_stats(USER,'MY_MV');

PL/SQL procedure successfully completed.

SQL> SELECT * FROM my_mv;
```

11. View and execute the **ws_05_22.sql** script. There is an intentional typographical error in the script to help identify the QUERY_OR_REWRITE hint at work.

```
SQL>  get ws_05_22.sql
   1    SELECT /*+ REWRITE_OR_ERROR */
   2         prod_id
   3  ,      avg(quantity_sold)
   4  FROM   sales
 5  WHERE  channel_id = 9
group  by prod_id;
SQL>@ws_05_22
```

**Student Observations:**

Full text match query rewrite is not possible because the QUANTITY_SOLD column does not appear in the underlying MY_MV materialized view definition.

12. Fix the error: Change QUANTITY_SOLD into AMOUNT_SOLD on line 3, and repeat the test. You can use the **ws_05_23.sql** script.

```
SQL>@ws_05_23
  1 SELECT /*+ REWRITE_OR_ERROR */
  2        prod_id
  3  ,     avg(amount_sold)
  4  FROM   sales
  5  WHERE  channel_id = 9
  6  group  by prod_id;
SQL> @ws_05_23
```

13. Run the **ws_05_24.sql** script to execute EXPLAIN PLAN against the query in the previous step and query the PLAN_TABLE table, to see the improved execution plan readability. Use **rp.sql** to view the execution plan.

```
SQL> get ws_05_24
  1  explain plan for
  2  SELECT prod_id
  3  ,      avg(amount_sold)
  4  FROM   sales
  5  WHERE  channel_id = 9
  6  group  by prod_id;
SQL>@ws_05_24
Explained.
SQL> @rp
```

**Student Observations:**

14. Before you can use the DBMS_MVIEW.EXPLAIN_REWRITE procedure, you must create the REWRITE_TABLE table with the **utlxrw.sql** script.

```
SQL>@utlxrw.sql
Table created
```

15. Using the **ws_05_25.sql** script, execute the DBMS_MVIEW.EXPLAIN_REWRITE procedure against the query and the MY_MV materialized view, and query the REWRITE_TABLE table to see the results.

```
SQL> get ws_05_25.sql
   1  execute dbms_mview.explain_rewrite -
   2  ( 'SELECT prod_id                   -
   3      ,      avg(amount_sold)         -
   4      FROM   sales                    -
   5      WHERE  channel_id = 9           -
   6      group  by prod_id'              -
   7  , 'SH.MY_MV'                        -
   8  , 'Practice 07-3'                   -
   9  ) ;
SQL> @ws_05_25
PL/SQL procedure successfully completed.
```

Use the **ws_05_26.sql** script to query the REWRITE_TABLE table.

```
SQL>get ws_05_26.sql
  1  SELECT message
  2  ,      original_cost, rewritten_cost
  3  FROM   rewrite_table;
SQL>@ws_05_26
```

**Student Observations:**

### **Part B**: **Using the SQLAccess Advisor to Recommend Materialized Views and Indexes**

To prepare the environment for using the SQLAccess Advisor, you perform the following steps. Materialized views and indexes can be present when the advisor is run, but for the purposes of this example they are removed so that you can see what the advisor will recommend. You need to also set up the cache so that the SQLAccess Advisor can generate some recommendations. Perform the following steps:

1. Open a terminal window. Change to workshop directory and log in to SQL*Plus as `system/oracle` and execute the following commands to clean up your environment:

```
    SQL> @prepare_for_advisor
```

2. Now you need to create the cache. Execute the following commands:

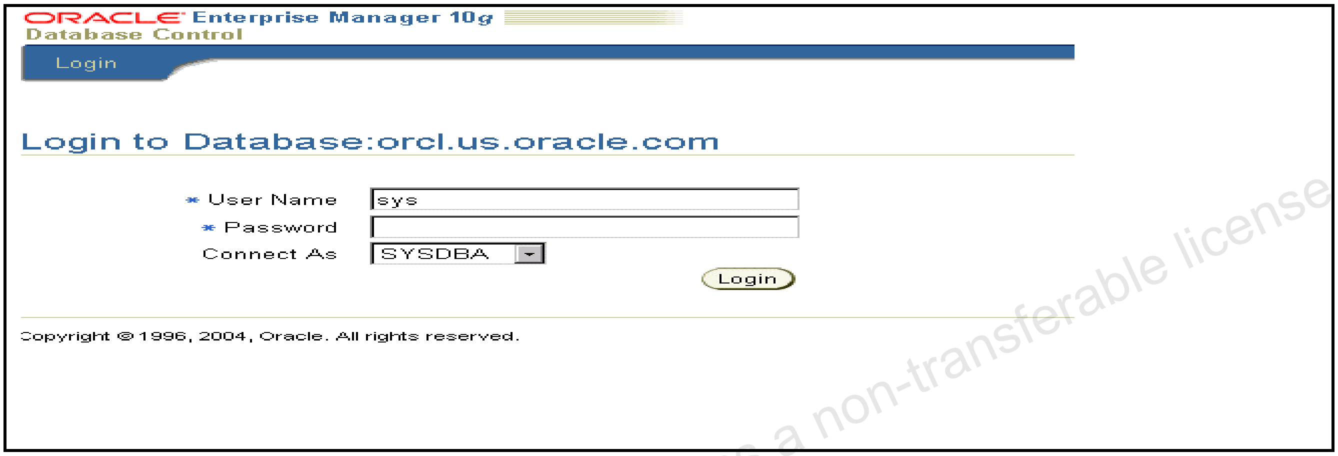
```
SQL> Get advisor_cache_setup
  1 alter system flush shared_pool;
  2  grant advisor to sh;
  3
  4  connect sh/sh;
  5 SELECT c.cust_last_name, sum(s.amount_sold) AS dollars,
  6 sum(s.quantity_sold) as quantity
  7 FROM sales s , customers c, products p
  8 WHERE c.cust_id = s.cust_id
  9 AND s.prod_id = p.prod_id
  10 AND c.cust_state_province IN ('Dublin','Galway')
  11 GROUP BY c.cust_last_name;
  12
  13 SELECT c.cust_id, SUM(amount_sold) AS dollar_sales
  14 FROM sales s, customers c WHERE s.cust_id= c.cust_id
  15 GROUP BY c.cust_id;
  16
  17 SELECT sum(unit_cost) FROM costs GROUP BY prod_id;

SQL> @advisor_cache_setup
```

### Using SQL Cache to Get Recommendations

You will use the SQL Cache you just set up to get some recommendations from the SQLAccess Advisor. Perform the following steps:

1. Open your browser and enter the following URL:

```
http://<hostname>:5500/em
Enter sys/<password> as SYSDBA and click Login.
```



ORACLE Enterprise Manager 10g
Database Control

Login

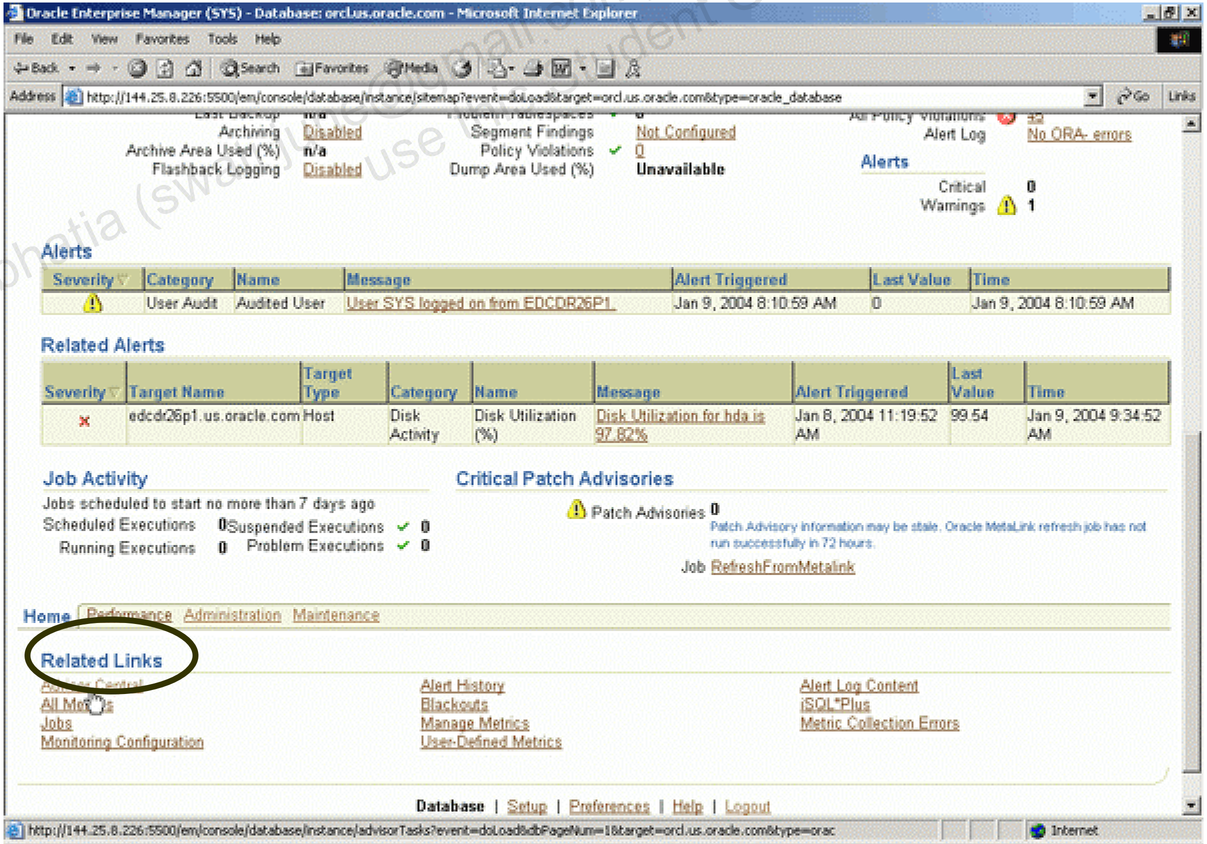
Login to Database:orcl.us.oracle.com

| | |
|---|---|
| * User Name | sys |
| * Password | |
| Connect As | SYSDBA |

(Login)

Copyright © 1996, 2004, Oracle. All rights reserved.

2. Scroll to the bottom of the **Home** page and click **Advisor Central** under **Related Links**.



Oracle Enterprise Manager (SYS) - Database: orcl.us.oracle.com - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address

http://144.25.8.226:5500/em/console/database/instance/sitemap?event=doLoad&target=orcl.us.oracle.com&type=oracle_database

| | | | | | |
|---|---|---|---|---|---|
| Archiving | Disabled | Segment Findings | Not Configured | Alert Log | No ORA- errors |
| Archive Area Used (%) | n/a | Policy Violations | ✓ 0 | | |
| Flashback Logging | Disabled | Dump Area Used (%) | Unavailable | | |

Alerts

| | |
|---|---|
| Critical | 0 |
| Warnings | ⚠ 1 |

Alerts

| Severity | Category | Name | Message | Alert Triggered | Last Value | Time |
|---|---|---|---|---|---|---|
| ⚠ | User Audit | Audited User | User SYS logged on from EDCDR26P1. | Jan 9, 2004 8:10:59 AM | 0 | Jan 9, 2004 8:10:59 AM |

Related Alerts

| Severity | Target Name | Target Type | Category | Name | Message | Alert Triggered | Last Value | Time |
|---|---|---|---|---|---|---|---|---|
| ✗ | edcdr26p1.us.oracle.com | Host | Disk Activity | Disk Utilization (%) | Disk Utilization for hda is 97.82% | Jan 8, 2004 11:19:52 AM | 99.54 | Jan 9, 2004 9:34:52 AM |

Job Activity

Jobs scheduled to start no more than 7 days ago

| | | | |
|---|---|---|---|
| Scheduled Executions | 0 | Suspended Executions | ✓ 0 |
| Running Executions | 0 | Problem Executions | ✓ 0 |

Critical Patch Advisories

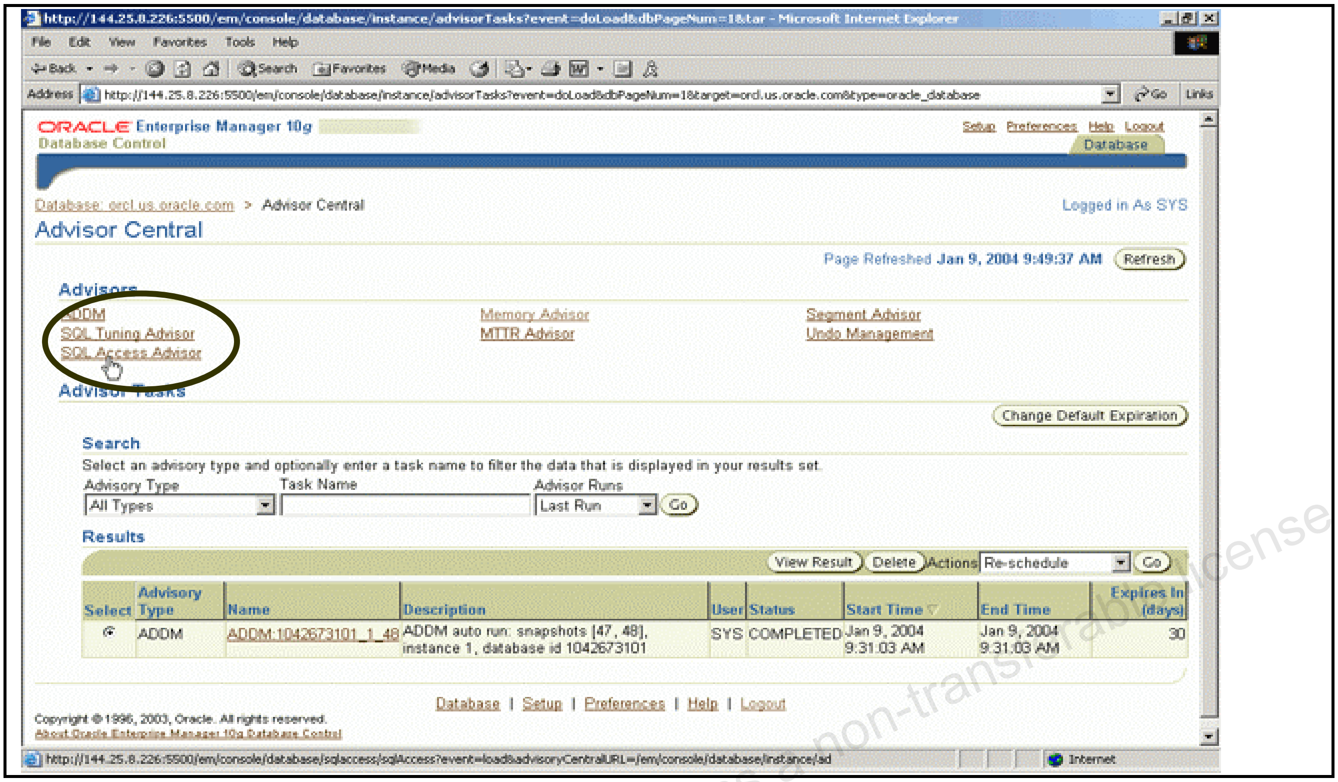
⚠ Patch Advisories 0
Patch Advisory information may be stale. Oracle MetaLink refresh job has not run successfully in 72 hours.
Job RefreshFromMetalink

Related Links

| | | |
|---|---|---|
| Advisor Central | Alert History | Alert Log Content |
| All Metrics | Blackouts | iSQL*Plus |
| Jobs | Manage Metrics | Metric Collection Errors |
| Monitoring Configuration | User-Defined Metrics | |

http://144.25.8.226:5500/em/console/database/instance/advisorTasks?event=doLoad&dbPageNum=1&target=orcl.us.oracle.com&type=orac

3. Click the **SQL Access Advisor** link.



http://144.25.8.226:5500/em/console/database/instance/advisorTasks?event=doLoad&dbPageNum=1&tar - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://144.25.8.226:5500/em/console/database/instance/advisorTasks?event=doLoad&dbPageNum=1&target=orcl.us.oracle.com&type=oracle_database
Go Links
ORACLE Enterprise Manager 10g
Database Control
Setup Preferences Help Logout
Database

Database: orcl.us.oracle.com > Advisor Central

Logged in As SYS

Advisor Central

Page Refreshed **Jan 9, 2004 9:49:37 AM** (Refresh)

**Advisors**

| | | |
|---|---|---|
| ADDM | Memory Advisor | Segment Advisor |
| SQL Tuning Advisor | MTTR Advisor | Undo Management |
| SQL Access Advisor | | |

**Advisor Tasks**

(Change Default Expiration)

**Search**

Select an advisory type and optionally enter a task name to filter the data that is displayed in your results set.

Advisory Type: All Types — Task Name: — Advisor Runs: Last Run (Go)

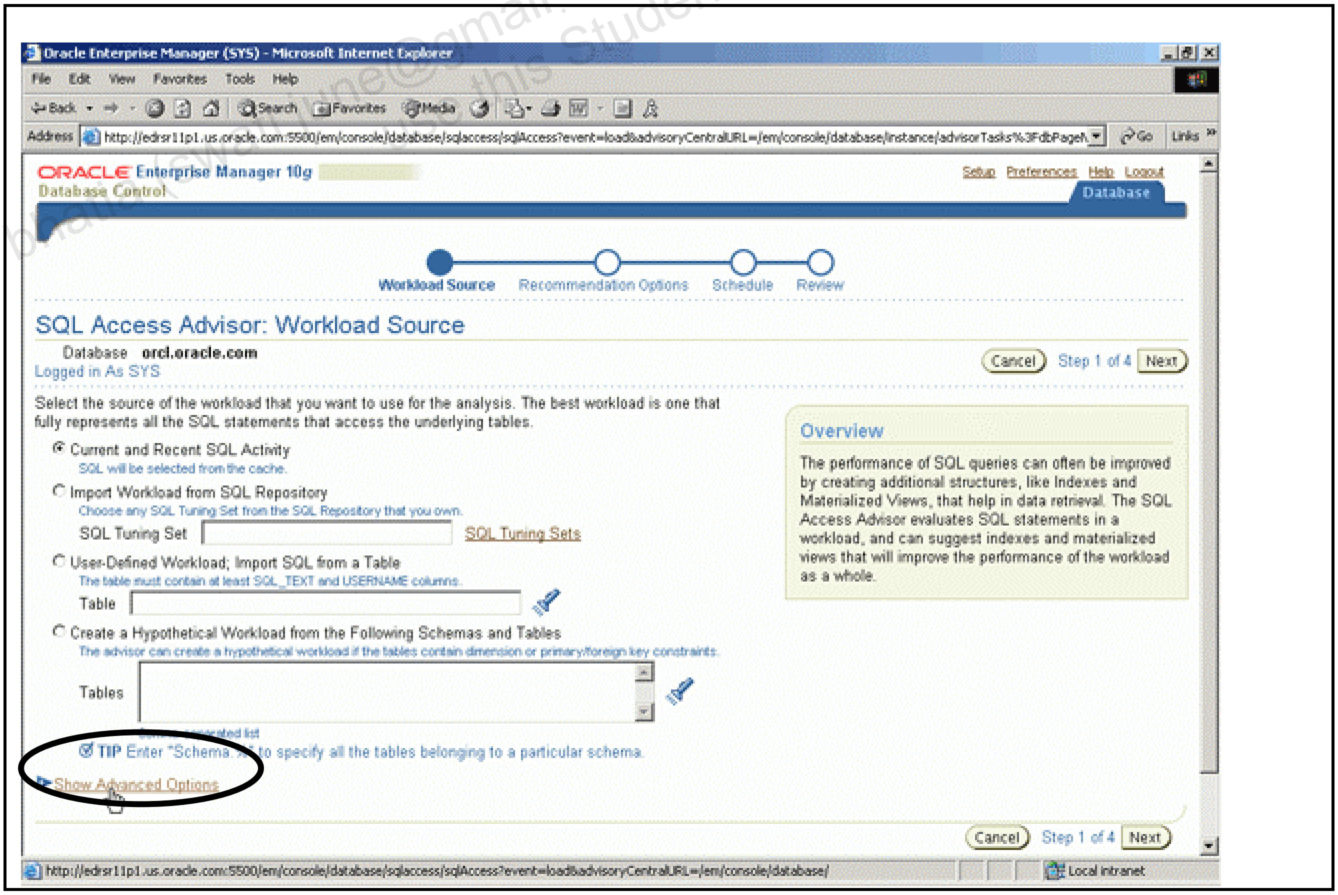
**Results**

(View Result) (Delete) Actions Re-schedule (Go)

| Select | Advisory Type | Name | Description | User | Status | Start Time | End Time | Expires In (days) |
|---|---|---|---|---|---|---|---|---|
| ● | ADDM | ADDM:1042673101_1_48 | ADDM auto run: snapshots [47, 48], instance 1, database id 1042673101 | SYS | COMPLETED | Jan 9, 2004 9:31:03 AM | Jan 9, 2004 9:31:03 AM | 30 |

Database | Setup | Preferences | Help | Logout
Copyright © 1996, 2003, Oracle. All rights reserved.
About Oracle Enterprise Manager 10g Database Control
http://144.25.8.226:5500/em/console/database/sqlaccess/sqlAccess?event=load&advisoryCentralURL=/em/console/database/instance/ad
Internet

4. Make sure **Current and Recent SQL Activity** is selected and click **Show Advanced Options**.



Oracle Enterprise Manager (SYS) - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://edrsr11p1.us.oracle.com:5500/em/console/database/sqlaccess/sqlAccess?event=load&advisoryCentralURL=/em/console/database/instance/advisorTasks%3FdbPageN
Go Links
ORACLE Enterprise Manager 10g
Database Control
Setup Preferences Help Logout
Database
Workload Source
Recommendation Options
Schedule
Review

SQL Access Advisor: Workload Source

Database **orcl.oracle.com**

Logged in As SYS

(Cancel) Step 1 of 4 (Next)

Select the source of the workload that you want to use for the analysis. The best workload is one that fully represents all the SQL statements that access the underlying tables.

● Current and Recent SQL Activity
SQL will be selected from the cache.

○ Import Workload from SQL Repository
Choose any SQL Tuning Set from the SQL Repository that you own.
SQL Tuning Set — SQL Tuning Sets

○ User-Defined Workload; Import SQL from a Table
The table must contain at least SQL_TEXT and USERNAME columns.
Table

○ Create a Hypothetical Workload from the Following Schemas and Tables
The advisor can create a hypothetical workload if the tables contain dimension or primary/foreign key constraints.
Tables
Comma-separated list

TIP Enter "Schema.%" to specify all the tables belonging to a particular schema.

Show Advanced Options

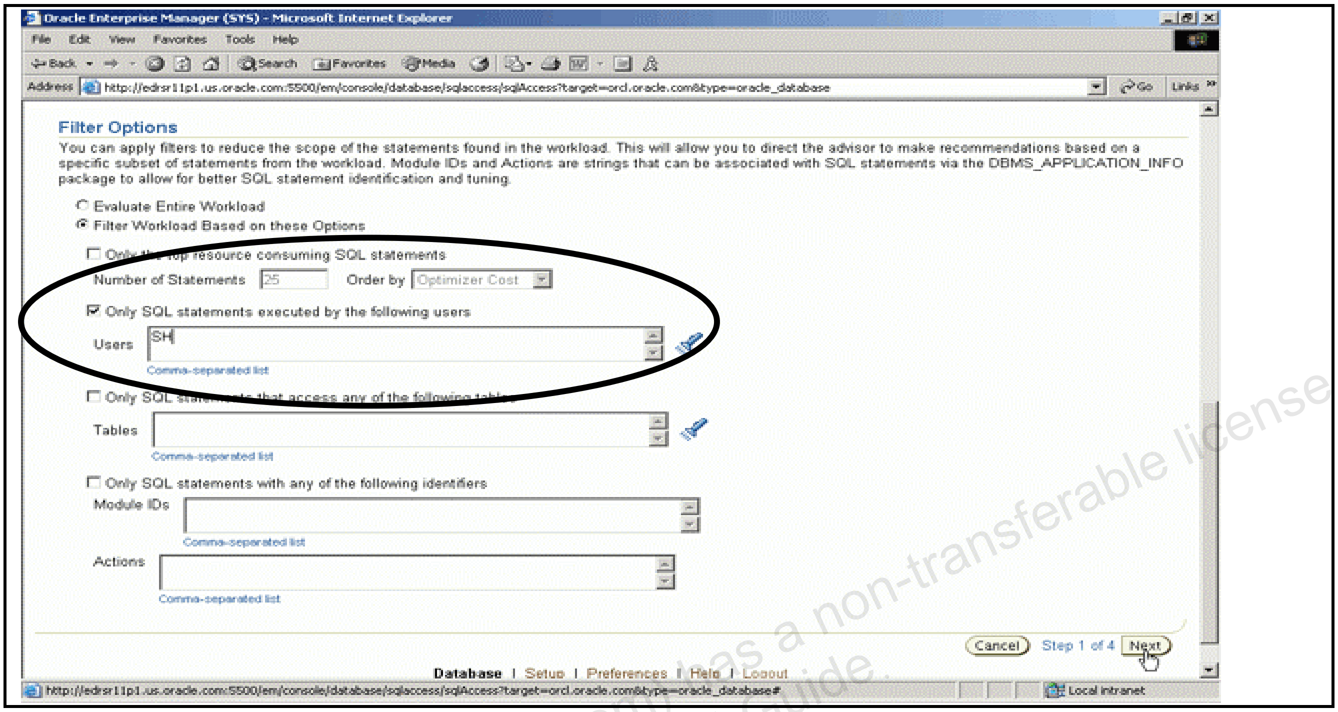
**Overview**

The performance of SQL queries can often be improved by creating additional structures, like Indexes and Materialized Views, that help in data retrieval. The SQL Access Advisor evaluates SQL statements in a workload, and can suggest indexes and materialized views that will improve the performance of the workload as a whole.

(Cancel) Step 1 of 4 (Next)

http://edrsr11p1.us.oracle.com:5500/em/console/database/sqlaccess/sqlAccess?event=load&advisoryCentralURL=/em/console/database/
Local intranet

5. Scroll down. Under **Filter Options**, select **Filter Workload Based on these Options** and select the filter named **Only SQL statements executed by the following users**. Then enter **`SH`** in the **Users** field and click **Next**.



Oracle Enterprise Manager (SYS) - Microsoft Internet Explorer

Address http://edrsr11p1.us.oracle.com:5500/em/console/database/sqlaccess/sqlAccess?target=orcl.oracle.com&type=oracle_database

**Filter Options**

You can apply filters to reduce the scope of the statements found in the workload. This will allow you to direct the advisor to make recommendations based on a specific subset of statements from the workload. Module IDs and Actions are strings that can be associated with SQL statements via the DBMS_APPLICATION_INFO package to allow for better SQL statement identification and tuning.

○ Evaluate Entire Workload
● Filter Workload Based on these Options

☐ Only the top resource consuming SQL statements
Number of Statements 25 Order by Optimizer Cost

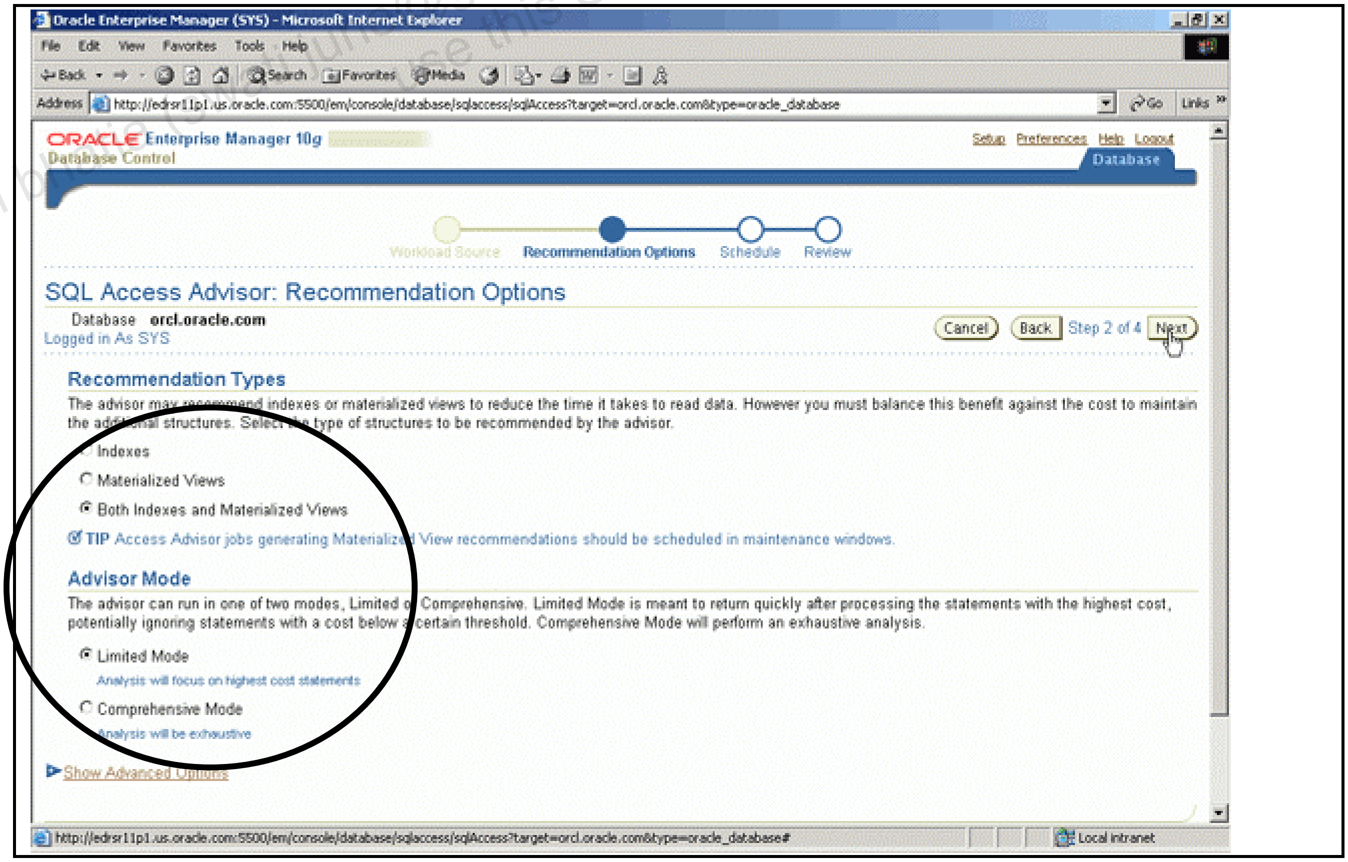
☑ Only SQL statements executed by the following users
Users SH
Comma-separated list

☐ Only SQL statements that access any of the following tables
Tables
Comma-separated list

☐ Only SQL statements with any of the following identifiers
Module IDs
Comma-separated list
Actions
Comma-separated list

File Edit View Favorites Tools Help
Back Search Favorites Media
Go Links
Cancel
Step 1 of 4
Next
Database | Setup | Preferences | Help | Logout
http://edrsr11p1.us.oracle.com:5500/em/console/database/sqlaccess/sqlAccess?target=orcl.oracle.com&type=oracle_database#
Local intranet

6. Click **Both Indexes and Materialized Views** and click **Next**.



Oracle Enterprise Manager (SYS) - Microsoft Internet Explorer

Address http://edrsr11p1.us.oracle.com:5500/em/console/database/sqlaccess/sqlAccess?target=orcl.oracle.com&type=oracle_database

**SQL Access Advisor: Recommendation Options**

Database **orcl.oracle.com**
Logged in As SYS

**Recommendation Types**

The advisor may recommend indexes or materialized views to reduce the time it takes to read data. However you must balance this benefit against the cost to maintain the additional structures. Select the type of structures to be recommended by the advisor.

○ Indexes
○ Materialized Views
● Both Indexes and Materialized Views

TIP Access Advisor jobs generating Materialized View recommendations should be scheduled in maintenance windows.

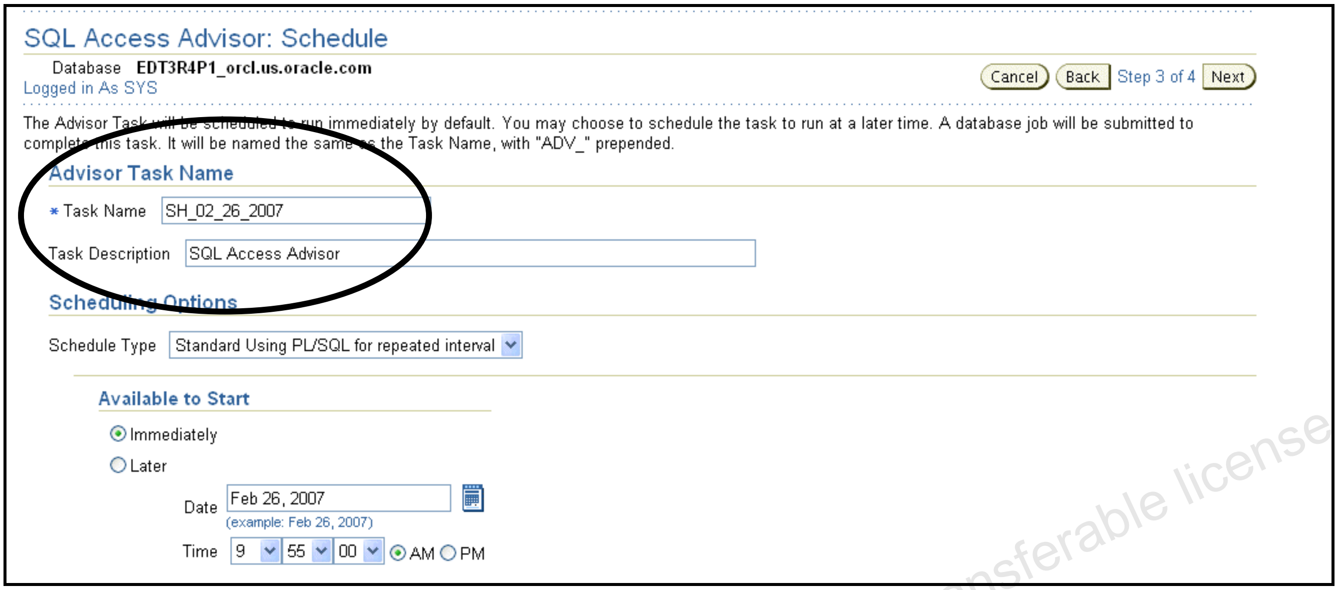
**Advisor Mode**

The advisor can run in one of two modes, Limited or Comprehensive. Limited Mode is meant to return quickly after processing the statements with the highest cost, potentially ignoring statements with a cost below a certain threshold. Comprehensive Mode will perform an exhaustive analysis.

● Limited Mode
Analysis will focus on highest cost statements
○ Comprehensive Mode
Analysis will be exhaustive

File Edit View Favorites Tools Help
Back Search Favorites Media
Go Links
ORACLE Enterprise Manager 10g
Database Control
Setup Preferences Help Logout
Database
Workload Source
Recommendation Options
Schedule
Review
Cancel
Back
Step 2 of 4
Next
Show Advanced Options
http://edrsr11p1.us.oracle.com:5500/em/console/database/sqlaccess/sqlAccess?target=orcl.oracle.com&type=oracle_database#
Local intranet

7. Enter the **Task Name** SH<*Today's Date*>, and then select "Standard Using PL/SQL for repeated interval" for the **Schedule Type**.



SQL Access Advisor: Schedule

Database EDT3R4P1_orcl.us.oracle.com
Logged in As SYS

The Advisor Task will be scheduled to run immediately by default. You may choose to schedule the task to run at a later time. A database job will be submitted to complete this task. It will be named the same as the Task Name, with "ADV_" prepended.

Advisor Task Name

* Task Name: SH_02_26_2007

Task Description: SQL Access Advisor

Scheduling Options

Schedule Type: Standard Using PL/SQL for repeated interval

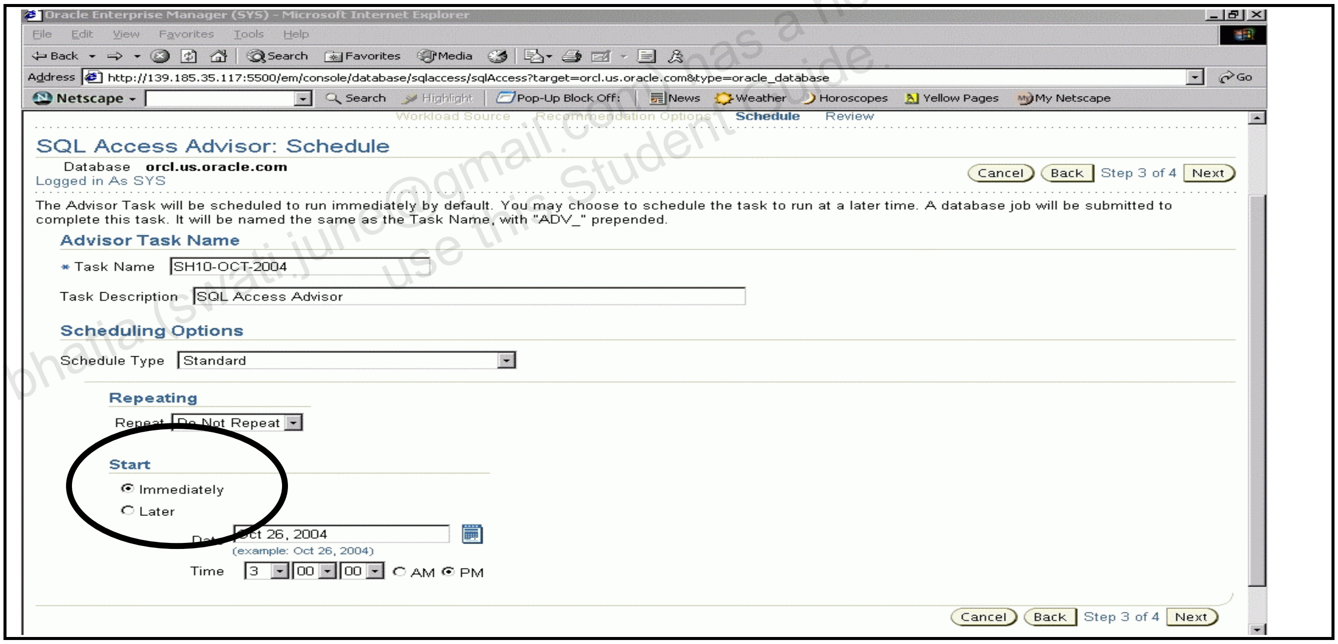
Available to Start

● Immediately
○ Later

Date: Feb 26, 2007 (example: Feb 26, 2007)

Time: 9 : 55 : 00 ● AM ○ PM

8. Under **Scheduling Options**, select **Start Immediately**, and click **Next**.



SQL Access Advisor: Schedule

Database orcl.us.oracle.com
Logged in As SYS

The Advisor Task will be scheduled to run immediately by default. You may choose to schedule the task to run at a later time. A database job will be submitted to complete this task. It will be named the same as the Task Name, with "ADV_" prepended.

Advisor Task Name

* Task Name: SH10-OCT-2004

Task Description: SQL Access Advisor

Scheduling Options

Schedule Type: Standard

Repeating

Repeat: Do Not Repeat

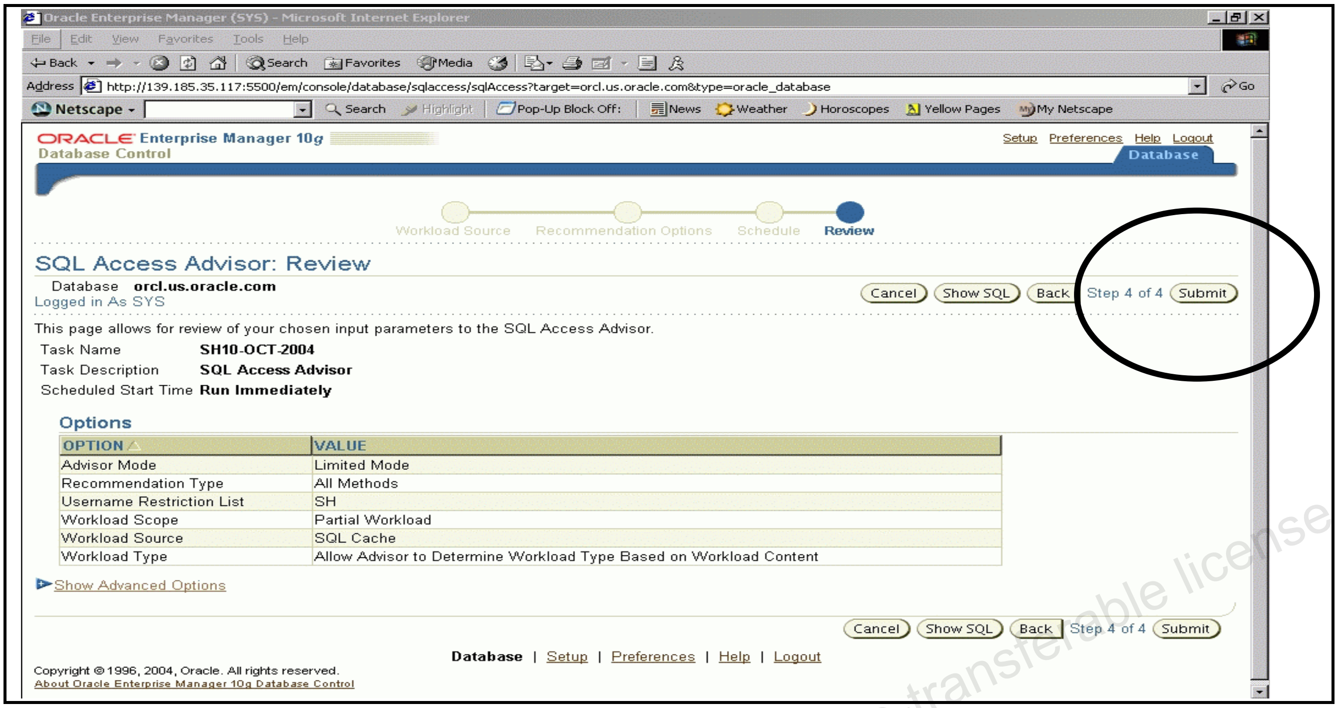
Start

● Immediately
○ Later

Date: Oct 26, 2004 (example: Oct 26, 2004)

Time: 3 : 00 : 00 ○ AM ● PM

9. In the **Summary** window, click **Submit**.



Oracle Enterprise Manager (SYS) - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://139.185.35.117:5500/em/console/database/sqlaccess/sqlAccess?target=orcl.us.oracle.com&type=oracle_database
Go
Netscape Search Highlight Pop-Up Block Off: News Weather Horoscopes Yellow Pages My Netscape
ORACLE Enterprise Manager 10g
Database Control
Setup Preferences Help Logout
Database
Workload Source
Recommendation Options
Schedule
Review

## SQL Access Advisor: Review

Database **orcl.us.oracle.com**
Logged in As SYS

Cancel
Show SQL
Back
Step 4 of 4
Submit

This page allows for review of your chosen input parameters to the SQL Access Advisor.

| | |
|---|---|
| Task Name | **SH10-OCT-2004** |
| Task Description | **SQL Access Advisor** |
| Scheduled Start Time | **Run Immediately** |

### Options

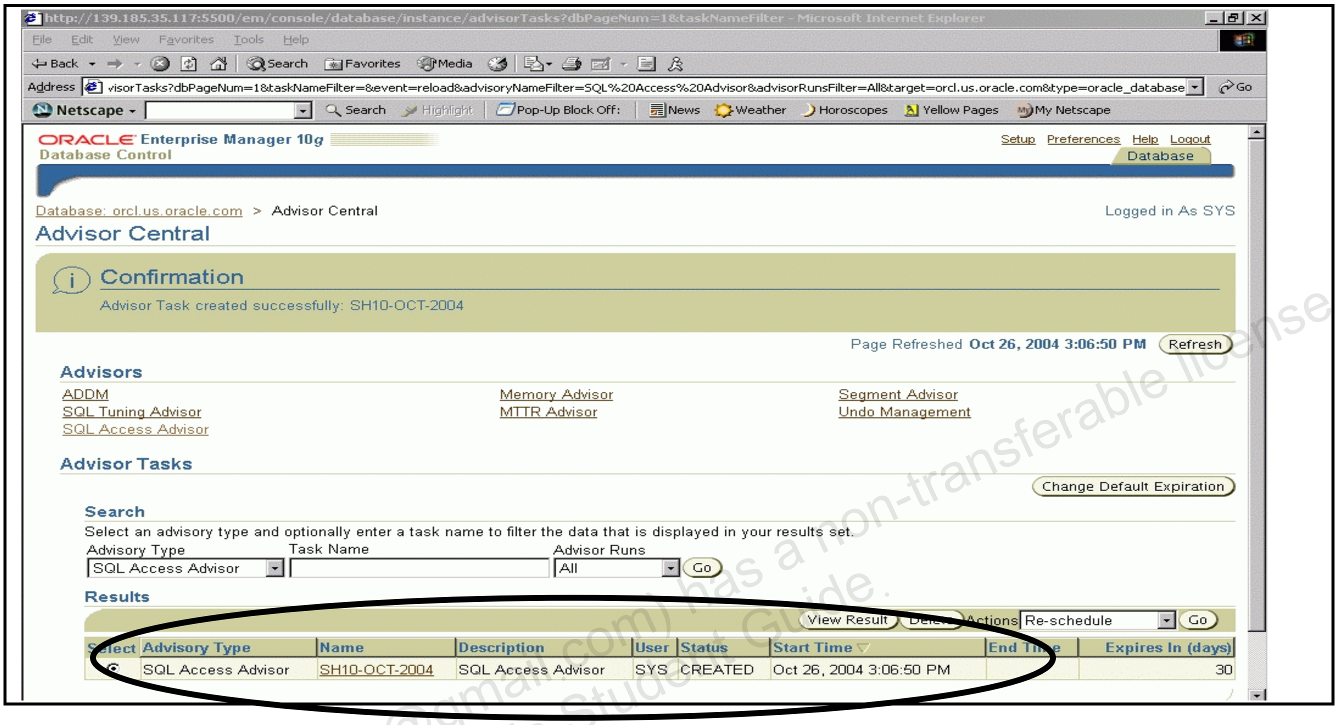
| OPTION △ | VALUE |
|---|---|
| Advisor Mode | Limited Mode |
| Recommendation Type | All Methods |
| Username Restriction List | SH |
| Workload Scope | Partial Workload |
| Workload Source | SQL Cache |
| Workload Type | Allow Advisor to Determine Workload Type Based on Workload Content |

Copyright © 1996, 2004, Oracle. All rights reserved.
About Oracle Enterprise Manager 10g Database Control

## Reviewing and Implementing the Recommendations

Now you can look at the results and implement them if you want. Perform the following steps:

1. Make sure your job is selected and click the **View Result** button.



http://139.185.35.117:5500/em/console/database/instance/advisorTasks?dbPageNum=1&taskNameFilter - Microsoft Internet Explorer

Address: visorTasks?dbPageNum=1&taskNameFilter=&event=reload&advisoryNameFilter=SQL%20Access%20Advisor&advisorRunsFilter=All&target=orcl.us.oracle.com&type=oracle_database

ORACLE Enterprise Manager 10*g* Database Control

Database: orcl.us.oracle.com > Advisor Central

Logged in As SYS

### Advisor Central

**Confirmation**

Advisor Task created successfully: SH10-OCT-2004

Page Refreshed **Oct 26, 2004 3:06:50 PM**

**Advisors**

ADDM
SQL Tuning Advisor
SQL Access Advisor
Memory Advisor
MTTR Advisor
Segment Advisor
Undo Management

**Advisor Tasks**

**Search**

Select an advisory type and optionally enter a task name to filter the data that is displayed in your results set.

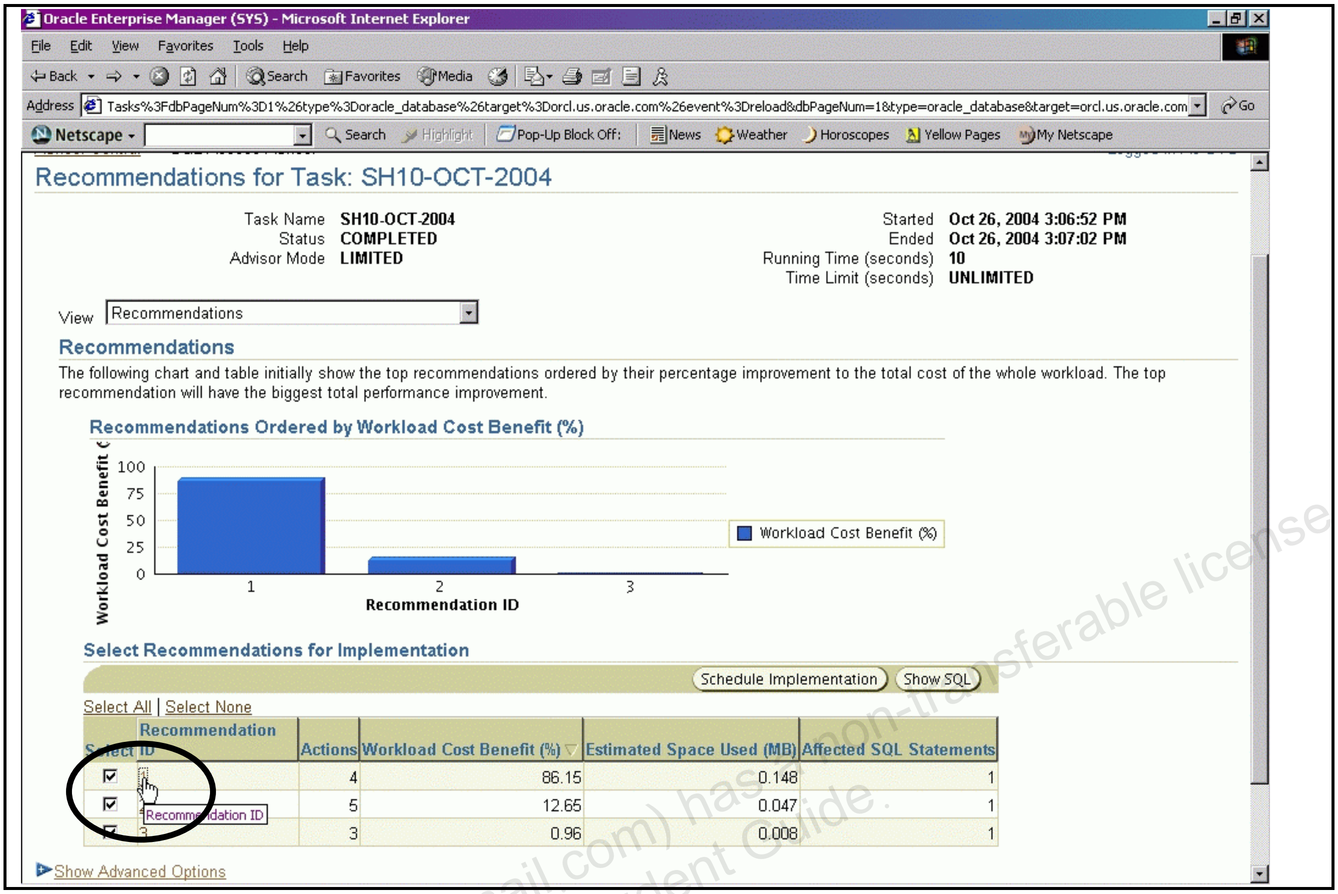
Advisory Type: SQL Access Advisor
Task Name:
Advisor Runs: All

**Results**

Actions: Re-schedule

| Select | Advisory Type | Name | Description | User | Status | Start Time | End Time | Expires In (days) |
|---|---|---|---|---|---|---|---|---|
| ● | SQL Access Advisor | SH10-OCT-2004 | SQL Access Advisor | SYS | CREATED | Oct 26, 2004 3:06:50 PM | | 30 |

2. Click the **Recommendation ID** 1 to see the details of the recommendations.


Oracle Enterprise Manager (SYS) - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address
Go
Netscape Search Highlight Pop-Up Block Off: News Weather Horoscopes Yellow Pages My Netscape

Address: Tasks%3FdbPageNum%3D1%26type%3Doracle_database%26target%3Dorcl.us.oracle.com%26event%3Dreload&dbPageNum=1&type=oracle_database&target=orcl.us.oracle.com

## Recommendations for Task: SH10-OCT-2004

| | | | |
|---|---|---|---|
| Task Name | SH10-OCT-2004 | Started | Oct 26, 2004 3:06:52 PM |
| Status | COMPLETED | Ended | Oct 26, 2004 3:07:02 PM |
| Advisor Mode | LIMITED | Running Time (seconds) | 10 |
| | | Time Limit (seconds) | UNLIMITED |

View: Recommendations

### Recommendations

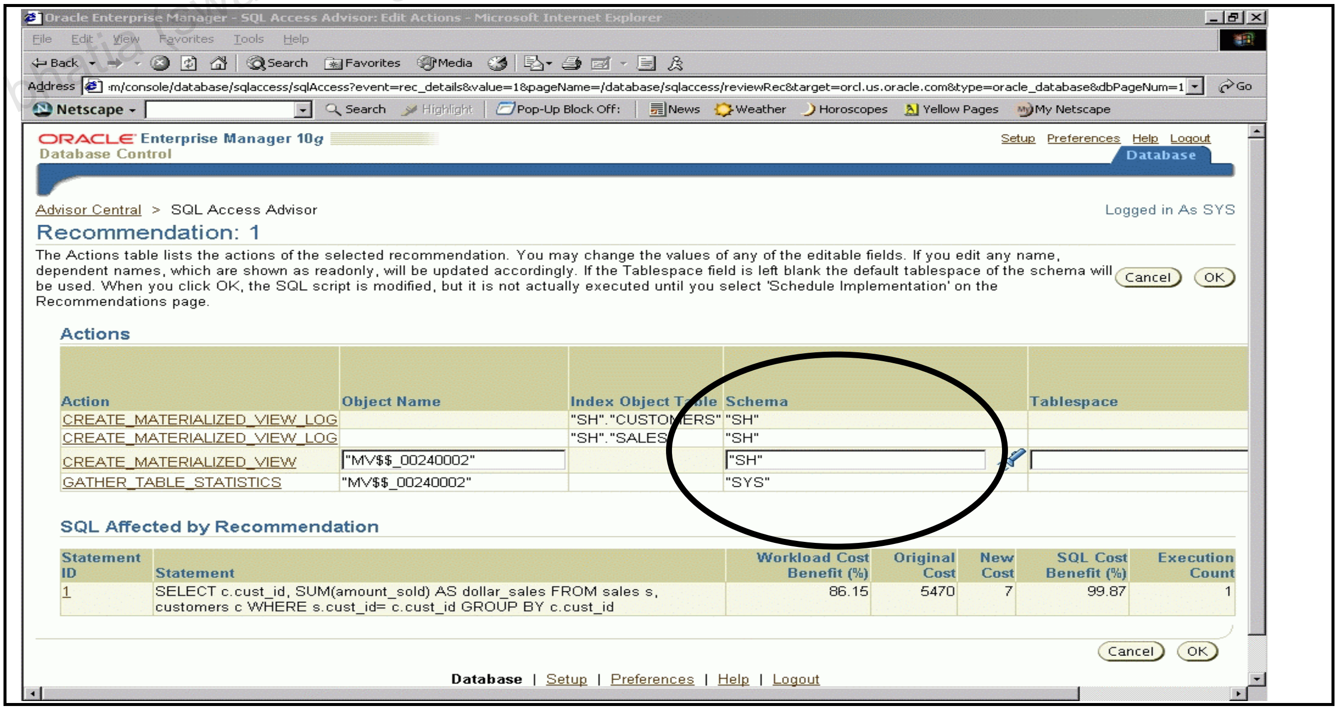
The following chart and table initially show the top recommendations ordered by their percentage improvement to the total cost of the whole workload. The top recommendation will have the biggest total performance improvement.

Recommendations Ordered by Workload Cost Benefit (%)

Select Recommendations for Implementation

| Select | Recommendation ID | Actions | Workload Cost Benefit (%) ▽ | Estimated Space Used (MB) | Affected SQL Statements |
|---|---|---|---|---|---|
| ☑ | 1 | 4 | 86.15 | 0.148 | 1 |
| ☑ | 2 | 5 | 12.65 | 0.047 | 1 |
| ☑ | 3 | 3 | 0.96 | 0.008 | 1 |

3. Here you can customize the **Object Name**, **Schema**, and **Tablespace** to implement the recommendations. Scroll down and change the **Schema Name** for the **Create Materialized View** to SH and click **OK**.


Oracle Enterprise Manager - SQL Access Advisor: Edit Actions - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address
Go
Netscape Search Highlight Pop-Up Block Off: News Weather Horoscopes Yellow Pages My Netscape
ORACLE Enterprise Manager 10g
Database Control
Setup Preferences Help Logout
Database
Cancel
OK
Cancel
OK
Database | Setup | Preferences | Help | Logout

Address: m/console/database/sqlaccess/sqlAccess?event=rec_details&value=1&pageName=/database/sqlaccess/reviewRec&target=orcl.us.oracle.com&type=oracle_database&dbPageNum=1

Advisor Central > SQL Access Advisor

Logged in As SYS

## Recommendation: 1

The Actions table lists the actions of the selected recommendation. You may change the values of any of the editable fields. If you edit any name, dependent names, which are shown as readonly, will be updated accordingly. If the Tablespace field is left blank the default tablespace of the schema will be used. When you click OK, the SQL script is modified, but it is not actually executed until you select 'Schedule Implementation' on the Recommendations page.

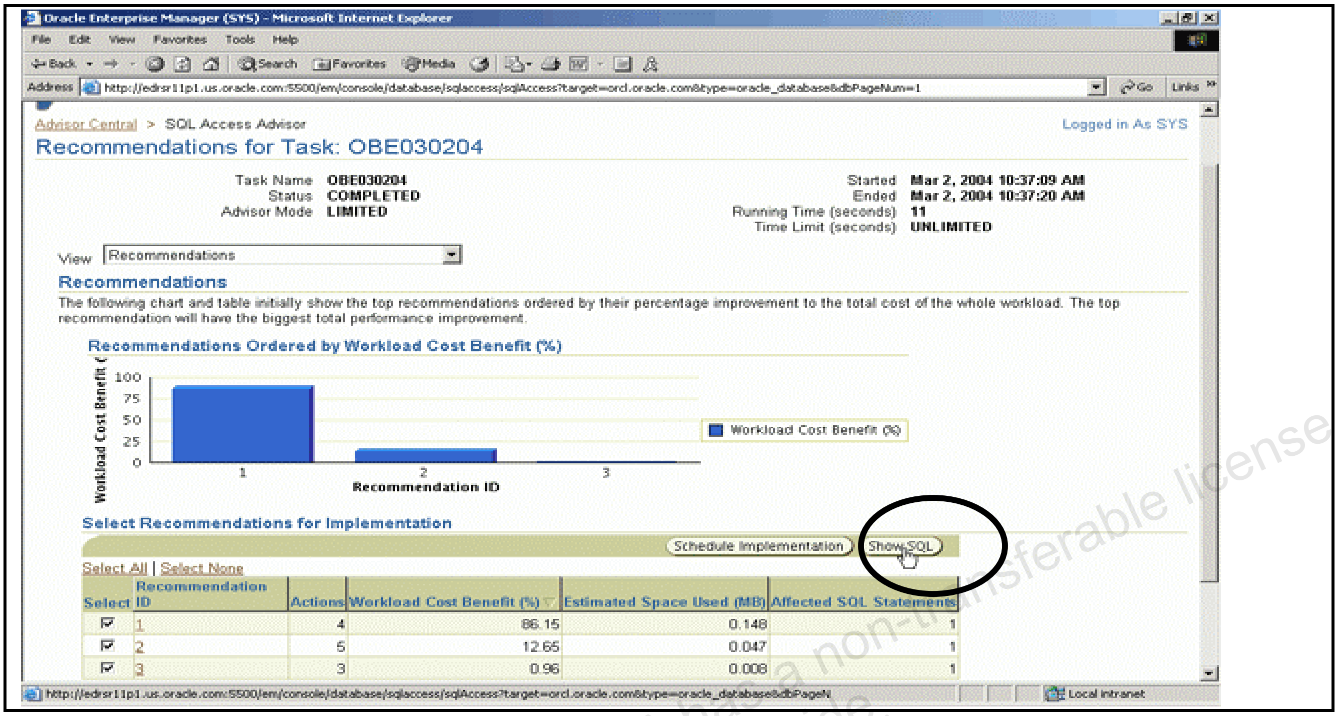
### Actions

| Action | Object Name | Index Object Table | Schema | Tablespace |
|---|---|---|---|---|
| CREATE_MATERIALIZED_VIEW_LOG | | "SH"."CUSTOMERS" | "SH" | |
| CREATE_MATERIALIZED_VIEW_LOG | | "SH"."SALES" | "SH" | |
| CREATE_MATERIALIZED_VIEW | "MV$$_00240002" | | "SH" | |
| GATHER_TABLE_STATISTICS | "MV$$_00240002" | | "SYS" | |

### SQL Affected by Recommendation

| Statement ID | Statement | Workload Cost Benefit (%) | Original Cost | New Cost | SQL Cost Benefit (%) | Execution Count |
|---|---|---|---|---|---|---|
| 1 | SELECT c.cust_id, SUM(amount_sold) AS dollar_sales FROM sales s, customers c WHERE s.cust_id= c.cust_id GROUP BY c.cust_id | 86.15 | 5470 | 7 | 99.87 | 1 |

4. To see the SQL script that will be executed when you schedule the implementation, click the **Show SQL** button.



Advisor Central > SQL Access Advisor

Logged in As SYS

Recommendations for Task: OBE030204

Task Name OBE030204
Status COMPLETED
Advisor Mode LIMITED

Started Mar 2, 2004 10:37:09 AM
Ended Mar 2, 2004 10:37:20 AM
Running Time (seconds) 11
Time Limit (seconds) UNLIMITED

View Recommendations

Recommendations

The following chart and table initially show the top recommendations ordered by their percentage improvement to the total cost of the whole workload. The top recommendation will have the biggest total performance improvement.

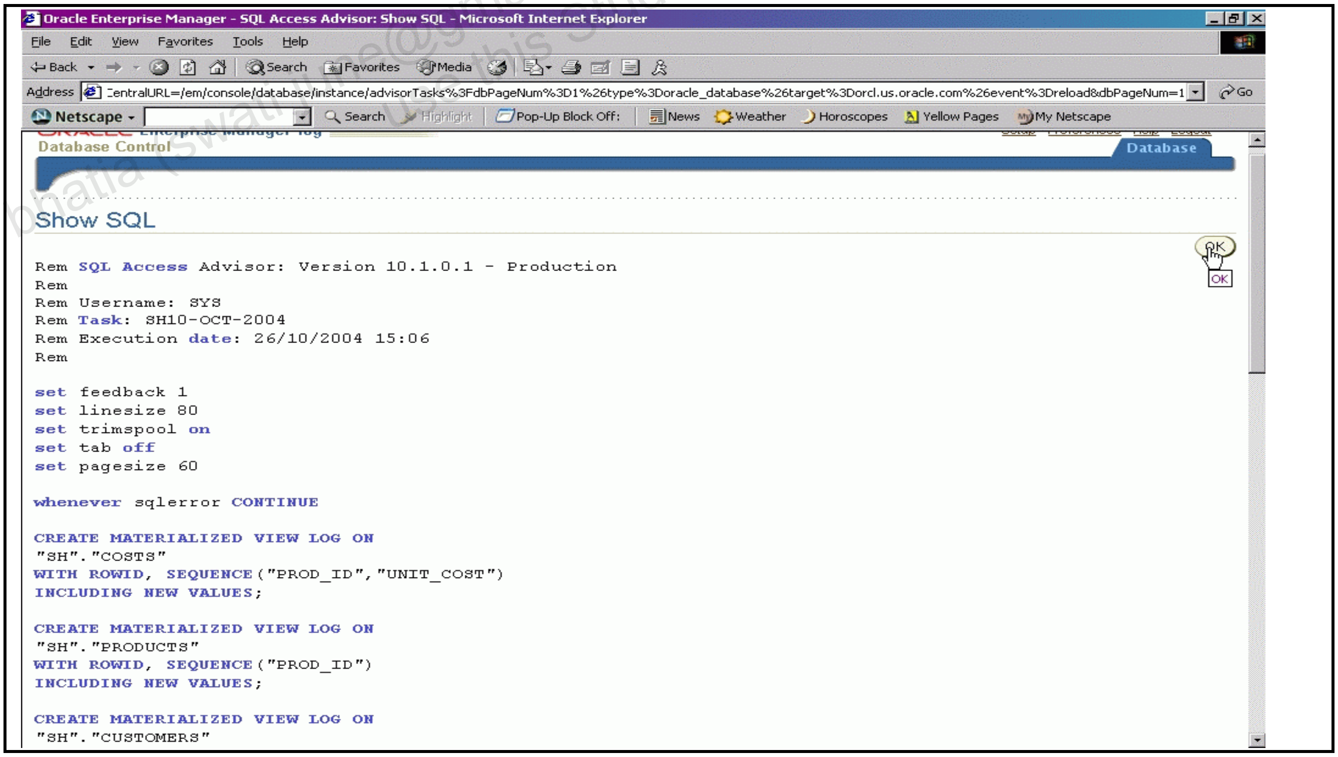
Oracle Enterprise Manager (SYS) - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://edrsr11p1.us.oracle.com:5500/em/console/database/sqlaccess/sqlAccess?target=orcl.oracle.com&type=oracle_database&dbPageNum=1
Recommendations Ordered by Workload Cost Benefit (%)
Workload Cost Benefit (
100
75
50
25
0
1
2
3
Recommendation ID
Workload Cost Benefit (%)
Schedule Implementation
Show SQL
Local intranet

Select Recommendations for Implementation

Select All | Select None

| Select | Recommendation ID | Actions | Workload Cost Benefit (%) | Estimated Space Used (MB) | Affected SQL Statements |
|---|---|---|---|---|---|
| ☑ | 1 | 4 | 86.15 | 0.148 | 1 |
| ☑ | 2 | 5 | 12.65 | 0.047 | 1 |
| ☑ | 3 | 3 | 0.96 | 0.008 | 1 |

5. Scroll down to the bottom and you will see the statements to create the materialized view with the change you just made. Click **OK**.

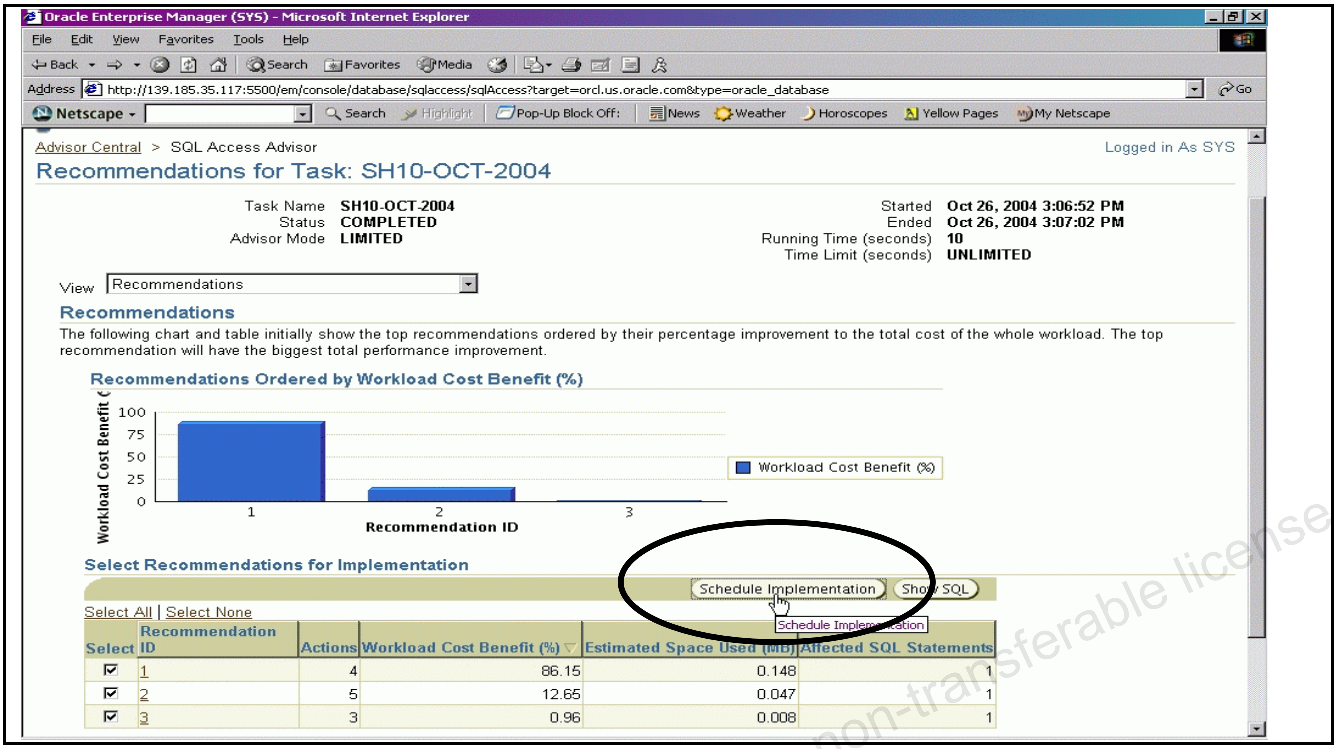


Oracle Enterprise Manager - SQL Access Advisor: Show SQL - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address centralURL=/em/console/database/instance/advisorTasks%3FdbPageNum%3D1%26type%3Doracle_database%26target%3Dorcl.us.oracle.com%26event%3Dreload&dbPageNum=1
Netscape
Database Control
Database
OK

Show SQL

```
Rem SQL Access Advisor: Version 10.1.0.1 - Production
Rem
Rem Username: SYS
Rem Task: SH10-OCT-2004
Rem Execution date: 26/10/2004 15:06
Rem

set feedback 1
set linesize 80
set trimspool on
set tab off
set pagesize 60

whenever sqlerror CONTINUE

CREATE MATERIALIZED VIEW LOG ON
"SH"."COSTS"
WITH ROWID, SEQUENCE("PROD_ID","UNIT_COST")
INCLUDING NEW VALUES;

CREATE MATERIALIZED VIEW LOG ON
"SH"."PRODUCTS"
WITH ROWID, SEQUENCE("PROD_ID")
INCLUDING NEW VALUES;

CREATE MATERIALIZED VIEW LOG ON
"SH"."CUSTOMERS"
```

6. To implement the recommendations, click the **Schedule Implementation** button.


Oracle Enterprise Manager (SYS) - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://139.185.35.117:5500/em/console/database/sqlaccess/sqlAccess?target=orcl.us.oracle.com&type=oracle_database
Advisor Central > SQL Access Advisor
Logged in As SYS

Recommendations for Task: SH10-OCT-2004

Task Name SH10-OCT-2004
Status COMPLETED
Advisor Mode LIMITED

Started Oct 26, 2004 3:06:52 PM
Ended Oct 26, 2004 3:07:02 PM
Running Time (seconds) 10
Time Limit (seconds) UNLIMITED

View Recommendations

Recommendations

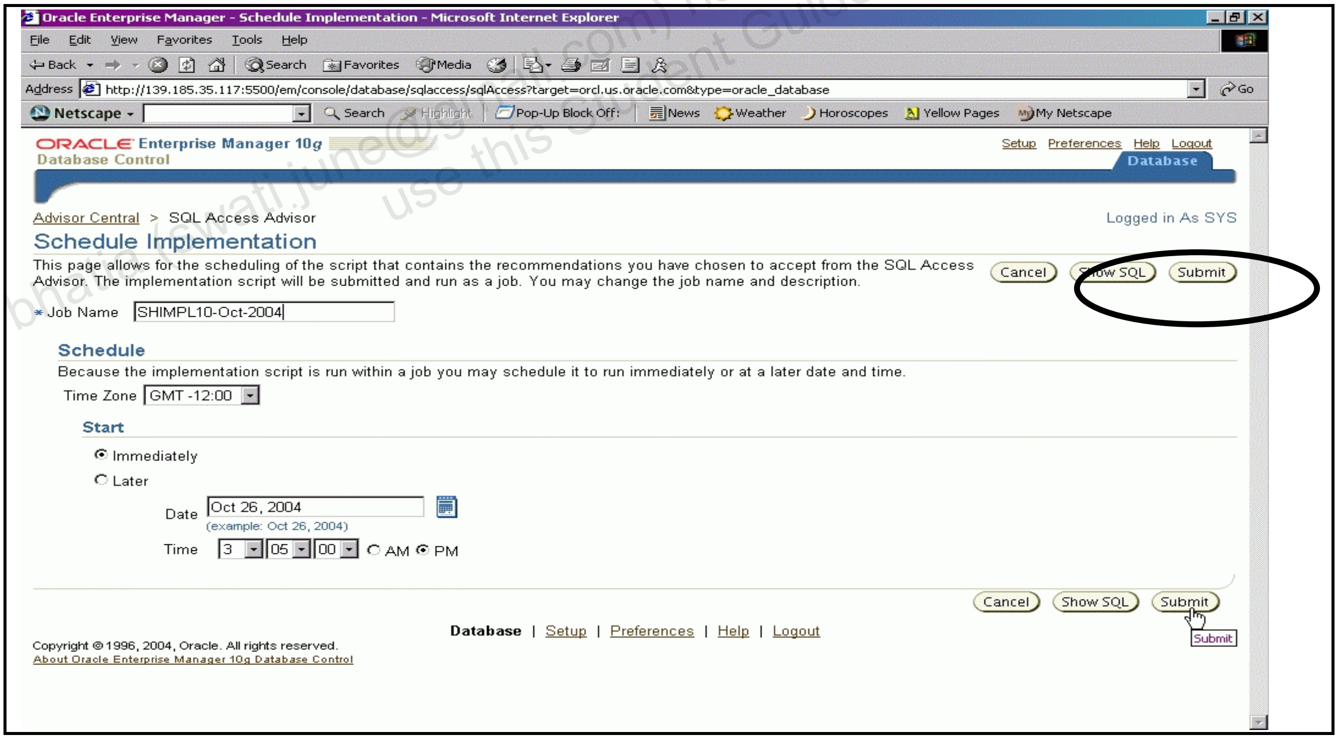
The following chart and table initially show the top recommendations ordered by their percentage improvement to the total cost of the whole workload. The top recommendation will have the biggest total performance improvement.

Recommendations Ordered by Workload Cost Benefit (%)

Select Recommendations for Implementation

| Select | Recommendation ID | Actions | Workload Cost Benefit (%) | Estimated Space Used (MB) | Affected SQL Statements |
|---|---|---|---|---|---|
| ☑ | 1 | 4 | 86.15 | 0.148 | 1 |
| ☑ | 2 | 5 | 12.65 | 0.047 | 1 |
| ☑ | 3 | 3 | 0.96 | 0.008 | 1 |

7. Enter SHIMPL<*today's date*> for the job name and click the **Submit** button.


Oracle Enterprise Manager - Schedule Implementation - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://139.185.35.117:5500/em/console/database/sqlaccess/sqlAccess?target=orcl.us.oracle.com&type=oracle_database
ORACLE Enterprise Manager 10g
Database Control
Setup Preferences Help Logout
Database
Advisor Central > SQL Access Advisor
Logged in As SYS

Schedule Implementation

This page allows for the scheduling of the script that contains the recommendations you have chosen to accept from the SQL Access Advisor. The implementation script will be submitted and run as a job. You may change the job name and description.

* Job Name SHIMPL10-Oct-2004

Schedule

Because the implementation script is run within a job you may schedule it to run immediately or at a later date and time.

Time Zone GMT -12:00

Start

● Immediately

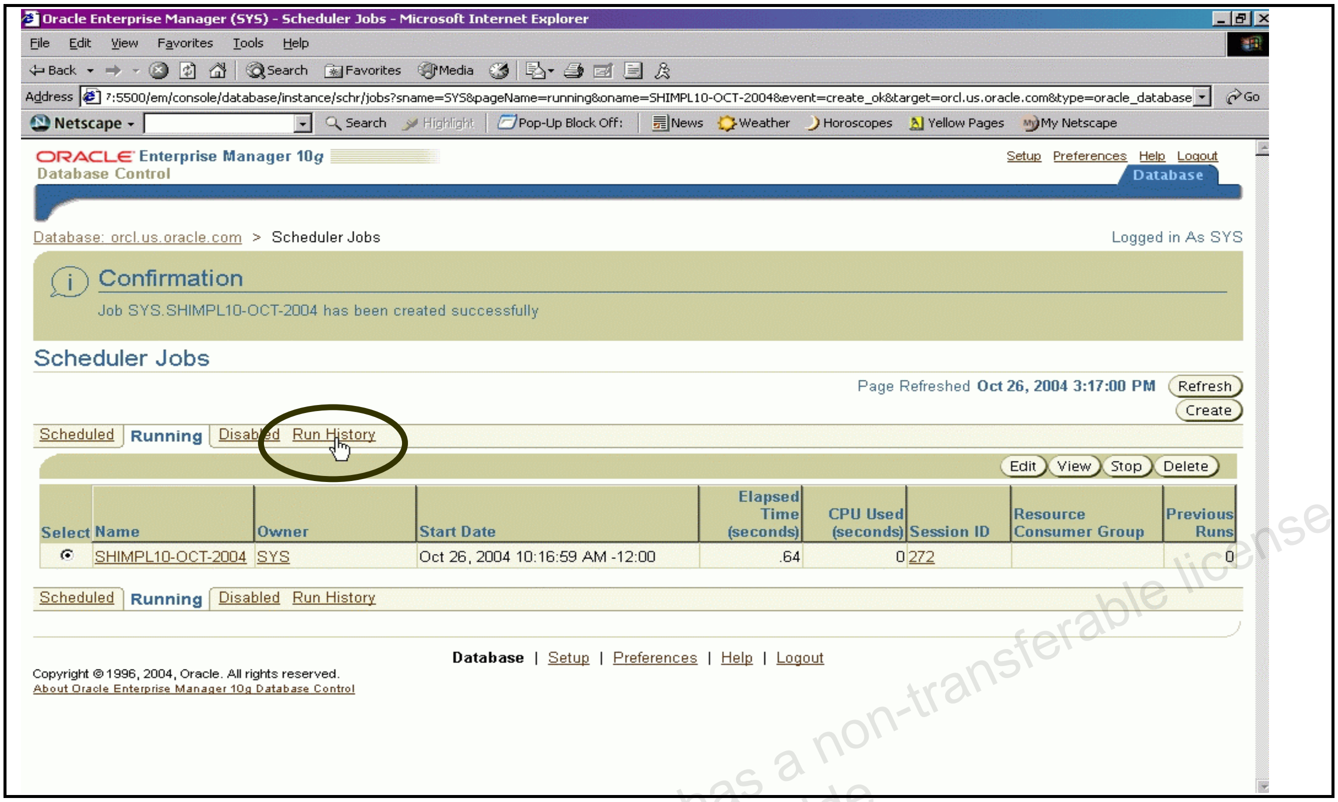
○ Later

Date Oct 26, 2004 (example: Oct 26, 2004)

Time 3 05 00 ○ AM ● PM

Copyright © 1996, 2004, Oracle. All rights reserved.
About Oracle Enterprise Manager 10g Database Control

8. Your implementation job was created and is now running. Click the **Run History** tab.


Oracle Enterprise Manager (SYS) - Scheduler Jobs - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address ?:5500/em/console/database/instance/schr/jobs?sname=SYS&pageName=running&oname=SHIMPL10-OCT-2004&event=create_ok&target=orcl.us.oracle.com&type=oracle_database
Go
Netscape Search Highlight Pop-Up Block Off: News Weather Horoscopes Yellow Pages My Netscape
ORACLE Enterprise Manager 10g
Database Control
Setup Preferences Help Logout
Database

Database: orcl.us.oracle.com > Scheduler Jobs

Logged in As SYS

Confirmation

Job SYS.SHIMPL10-OCT-2004 has been created successfully

Scheduler Jobs

Page Refreshed Oct 26, 2004 3:17:00 PM

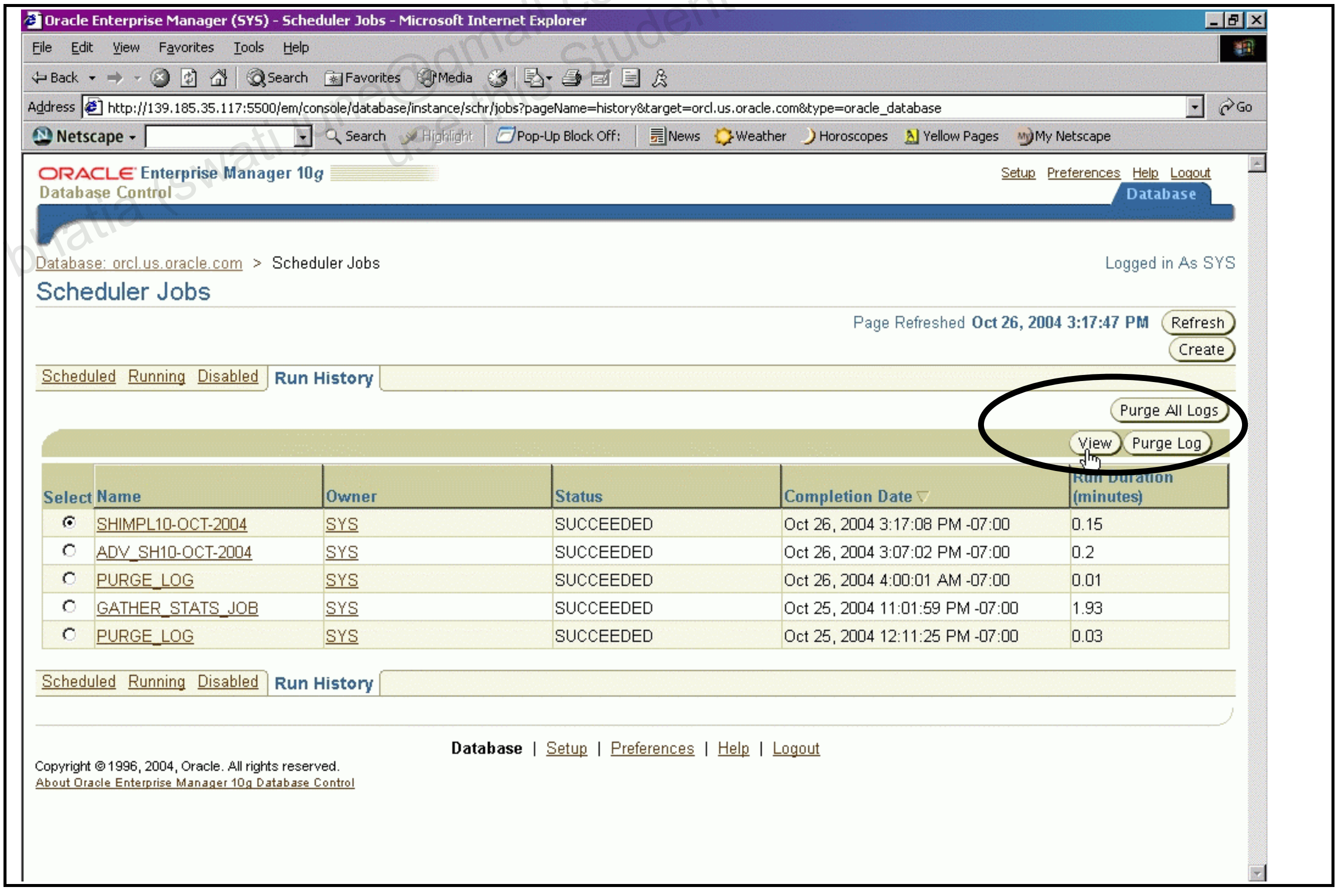
Refresh
Create
Scheduled Running Disabled Run History
Edit View Stop Delete

| Select | Name | Owner | Start Date | Elapsed Time (seconds) | CPU Used (seconds) | Session ID | Resource Consumer Group | Previous Runs |
|---|---|---|---|---|---|---|---|---|
| ● | SHIMPL10-OCT-2004 | SYS | Oct 26, 2004 10:16:59 AM -12:00 | .64 | 0 | 272 | | 0 |

Scheduled Running Disabled Run History
Database | Setup | Preferences | Help | Logout

Copyright © 1996, 2004, Oracle. All rights reserved.
About Oracle Enterprise Manager 10g Database Control

9. Make sure your job is selected, and then click **View**.


Oracle Enterprise Manager (SYS) - Scheduler Jobs - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://139.185.35.117:5500/em/console/database/instance/schr/jobs?pageName=history&target=orcl.us.oracle.com&type=oracle_database
Go
Netscape Search Highlight Pop-Up Block Off: News Weather Horoscopes Yellow Pages My Netscape
ORACLE Enterprise Manager 10g
Database Control
Setup Preferences Help Logout
Database

Database: orcl.us.oracle.com > Scheduler Jobs

Logged in As SYS

Scheduler Jobs

Page Refreshed Oct 26, 2004 3:17:47 PM

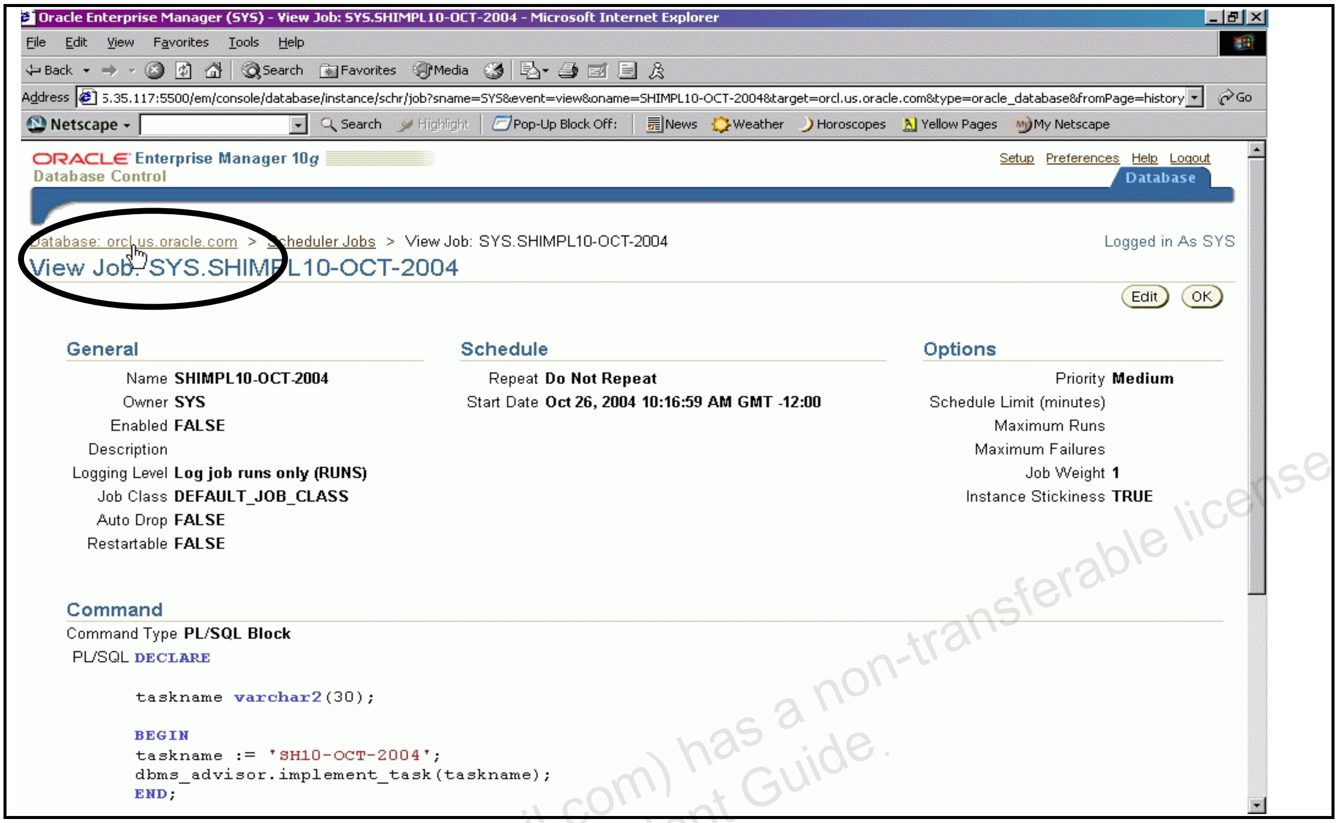
Refresh
Create
Scheduled Running Disabled Run History
Purge All Logs
View Purge Log

| Select | Name | Owner | Status | Completion Date ▽ | Run Duration (minutes) |
|---|---|---|---|---|---|
| ● | SHIMPL10-OCT-2004 | SYS | SUCCEEDED | Oct 26, 2004 3:17:08 PM -07:00 | 0.15 |
| ○ | ADV_SH10-OCT-2004 | SYS | SUCCEEDED | Oct 26, 2004 3:07:02 PM -07:00 | 0.2 |
| ○ | PURGE_LOG | SYS | SUCCEEDED | Oct 26, 2004 4:00:01 AM -07:00 | 0.01 |
| ○ | GATHER_STATS_JOB | SYS | SUCCEEDED | Oct 25, 2004 11:01:59 PM -07:00 | 1.93 |
| ○ | PURGE_LOG | SYS | SUCCEEDED | Oct 25, 2004 12:11:25 PM -07:00 | 0.03 |

Scheduled Running Disabled Run History
Database | Setup | Preferences | Help | Logout

Copyright © 1996, 2004, Oracle. All rights reserved.
About Oracle Enterprise Manager 10g Database Control

10. Review the summary and click your **Database** breadcrumb.


Oracle Enterprise Manager (SYS) - View Job: SYS.SHIMPL10-OCT-2004 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address 5.35.117:5500/em/console/database/instance/schr/job?sname=SYS&event=view&oname=SHIMPL10-OCT-2004&target=orcl.us.oracle.com&type=oracle_database&fromPage=history
Go
Netscape Search Highlight Pop-Up Block Off: News Weather Horoscopes Yellow Pages My Netscape
ORACLE Enterprise Manager 10g
Database Control
Setup Preferences Help Logout
Database
Edit
OK

Database: orcl.us.oracle.com > Scheduler Jobs > View Job: SYS.SHIMPL10-OCT-2004

Logged in As SYS

View Job: SYS.SHIMPL10-OCT-2004

General

Name SHIMPL10-OCT-2004
Owner SYS
Enabled FALSE
Description
Logging Level Log job runs only (RUNS)
Job Class DEFAULT_JOB_CLASS
Auto Drop FALSE
Restartable FALSE

Schedule

Repeat Do Not Repeat
Start Date Oct 26, 2004 10:16:59 AM GMT -12:00

Options

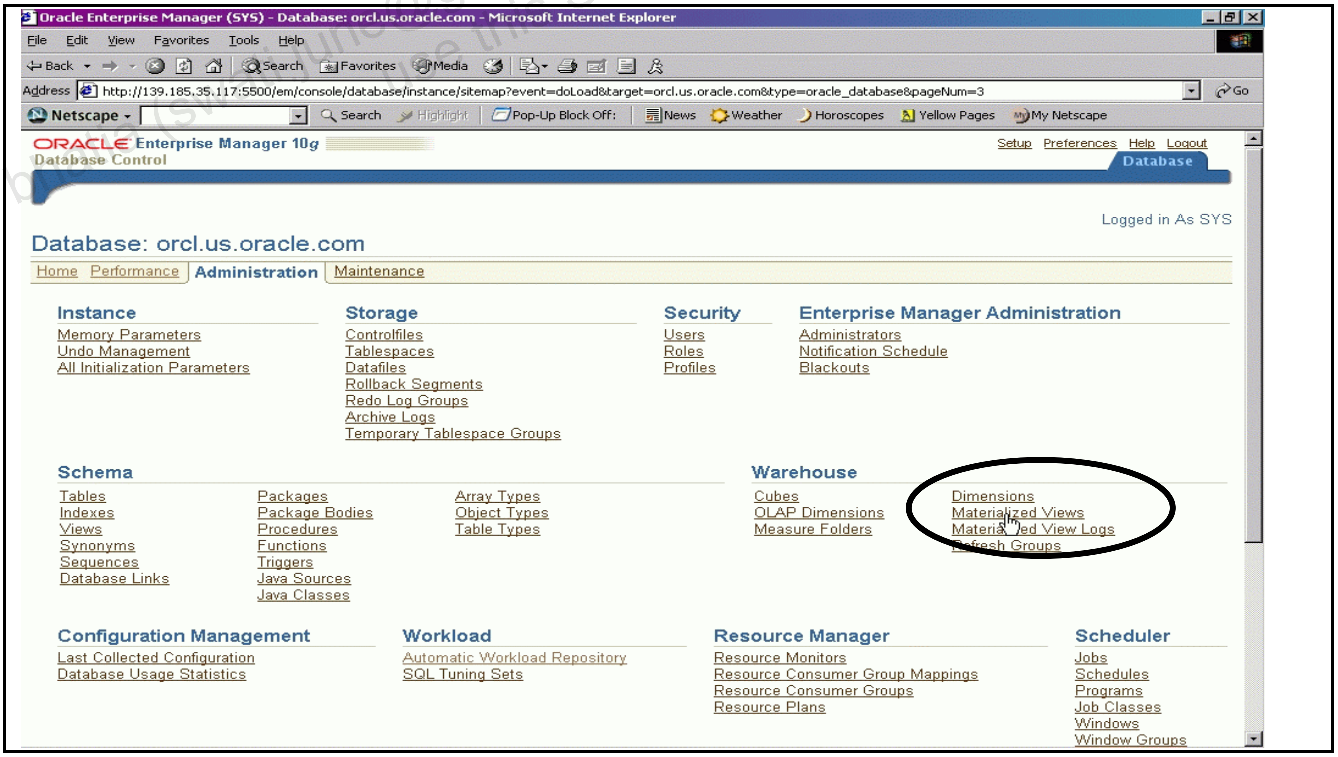
Priority Medium
Schedule Limit (minutes)
Maximum Runs
Maximum Failures
Job Weight 1
Instance Stickiness TRUE

Command

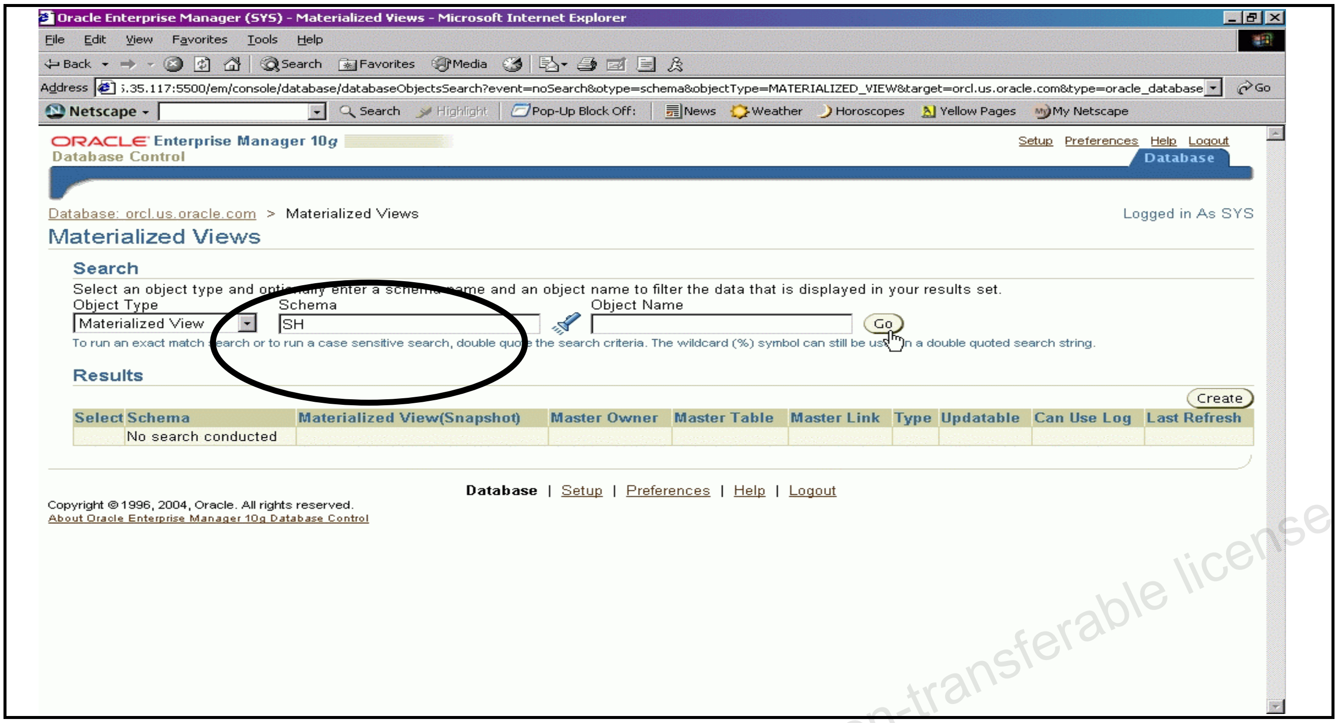
Command Type PL/SQL Block

```
PL/SQL DECLARE

      taskname varchar2(30);

      BEGIN
      taskname := 'SH10-OCT-2004';
      dbms_advisor.implement_task(taskname);
      END;
```

11. Click **Materialized Views** from the **Administration** tab.


Oracle Enterprise Manager (SYS) - Database: orcl.us.oracle.com - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://139.185.35.117:5500/em/console/database/instance/sitemap?event=doLoad&target=orcl.us.oracle.com&type=oracle_database&pageNum=3
Go
Netscape Search Highlight Pop-Up Block Off: News Weather Horoscopes Yellow Pages My Netscape
ORACLE Enterprise Manager 10g
Database Control
Setup Preferences Help Logout
Database
Logged in As SYS
Database: orcl.us.oracle.com
Home Performance Administration Maintenance
Instance
Memory Parameters
Undo Management
All Initialization Parameters
Storage
Controlfiles
Tablespaces
Datafiles
Rollback Segments
Redo Log Groups
Archive Logs
Temporary Tablespace Groups
Security
Users
Roles
Profiles
Enterprise Manager Administration
Administrators
Notification Schedule
Blackouts
Schema
Tables
Indexes
Views
Synonyms
Sequences
Database Links
Packages
Package Bodies
Procedures
Functions
Triggers
Java Sources
Java Classes
Array Types
Object Types
Table Types
Warehouse
Cubes
OLAP Dimensions
Measure Folders
Dimensions
Materialized Views
Materialized View Logs
Refresh Groups
Configuration Management
Last Collected Configuration
Database Usage Statistics
Workload
Automatic Workload Repository
SQL Tuning Sets
Resource Manager
Resource Monitors
Resource Consumer Group Mappings
Resource Consumer Groups
Resource Plans
Scheduler
Jobs
Schedules
Programs
Job Classes
Windows
Window Groups

12. Enter **`SH`** in the **Schema** field and click **Go**.



Oracle Enterprise Manager (SYS) - Materialized Views - Microsoft Internet Explorer

Address: ;.35.117:5500/em/console/database/databaseObjectsSearch?event=noSearch&otype=schema&objectType=MATERIALIZED_VIEW&target=orcl.us.oracle.com&type=oracle_database

Database: orcl.us.oracle.com > Materialized Views

Logged in As SYS

Materialized Views

Search

Select an object type and optionally enter a schema name and an object name to filter the data that is displayed in your results set.

Object Type: Materialized View

Schema: SH

Object Name:

To run an exact match search or to run a case sensitive search, double quote the search criteria. The wildcard (%) symbol can still be used in a double quoted search string.

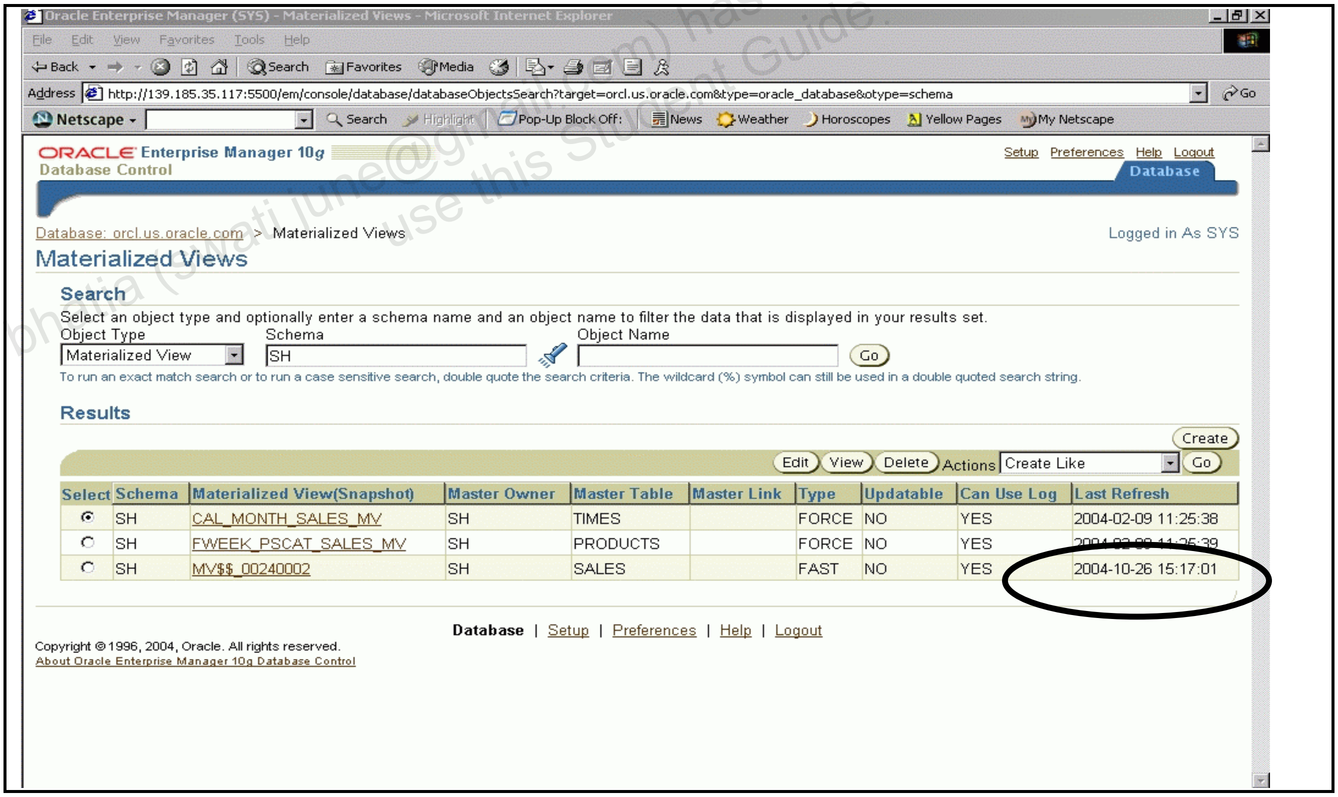
Results

| Select | Schema | Materialized View(Snapshot) | Master Owner | Master Table | Master Link | Type | Updatable | Can Use Log | Last Refresh |
|---|---|---|---|---|---|---|---|---|---|
| | No search conducted | | | | | | | | |

Copyright © 1996, 2004, Oracle. All rights reserved.
About Oracle Enterprise Manager 10g Database Control

File Edit View Favorites Tools Help
Back Search Favorites Media
Netscape Search Highlight Pop-Up Block Off: News Weather Horoscopes Yellow Pages My Netscape
ORACLE Enterprise Manager 10g
Database Control
Setup Preferences Help Logout
Database
Go
Create
Database | Setup | Preferences | Help | Logout

13. Note that the newly created materialized view appears in the list.



Oracle Enterprise Manager (SYS) - Materialized Views - Microsoft Internet Explorer

Address: http://139.185.35.117:5500/em/console/database/databaseObjectsSearch?target=orcl.us.oracle.com&type=oracle_database&otype=schema

Database: orcl.us.oracle.com > Materialized Views

Logged in As SYS

Materialized Views

Search

Select an object type and optionally enter a schema name and an object name to filter the data that is displayed in your results set.

Object Type: Materialized View

Schema: SH

Object Name:

To run an exact match search or to run a case sensitive search, double quote the search criteria. The wildcard (%) symbol can still be used in a double quoted search string.

Results

Actions: Create Like

| Select | Schema | Materialized View(Snapshot) | Master Owner | Master Table | Master Link | Type | Updatable | Can Use Log | Last Refresh |
|---|---|---|---|---|---|---|---|---|---|
| ● | SH | CAL_MONTH_SALES_MV | SH | TIMES | | FORCE | NO | YES | 2004-02-09 11:25:38 |
| ○ | SH | FWEEK_PSCAT_SALES_MV | SH | PRODUCTS | | FORCE | NO | YES | 2004-02-09 11:25:39 |
| ○ | SH | MV$$_00240002 | SH | SALES | | FAST | NO | YES | 2004-10-26 15:17:01 |

Copyright © 1996, 2004, Oracle. All rights reserved.
About Oracle Enterprise Manager 10g Database Control

File Edit View Favorites Tools Help
Back Search Favorites Media
Netscape Search Highlight Pop-Up Block Off: News Weather Horoscopes Yellow Pages My Netscape
ORACLE Enterprise Manager 10g
Database Control
Setup Preferences Help Logout
Database
Go
Create
Edit View Delete
Database | Setup | Preferences | Help | Logout

14. Close your browser.

## Workshop 6: Using Hints

### Objective:

Using hints to change the:

- Access paths
- Join methods

This workshop can be done entirely in the SQL*Plus environment. Try to work in small groups and discuss the workshop results. Each time you load a new SQL statement, try to predict what the optimizer will do before running the statement. Take notes during the workshop as an aid for the wrap-up discussion.

1. Open a terminal window. Change to workshop directory. Start SQL*Plus. Log in as **SH/SH**. Drop all indexes on the SALES table. Analyze the SQL statement in **ws_06_01.sql** by using SQL*TRACE set to TRACE ONLY.

```
SQL> @dai
On which table : sales
SQL>@atto
SQL> get ws_06_01
  1  SELECT c.cust_last_name
  2  FROM customers c
  3  WHERE c.country_id = 52790
  4  AND c.cust_id NOT IN (SELECT s.cust_id
 5*  FROM sales s)
SQL> @ws_06_01
SQL> @atoff
```

This is a SELECT statement with a subquery. Because the predicate with the subquery contains a NOT IN operator, this statement is also known as an *anti-join*.

By specifying hints to use a sort/merge or a hash operation instead (by using the MERGE_AJ or HASH_AJ hints in the subquery), there is a good chance of improving the performance of the statement. Try to find the optimal performance for this query by specifying hints. You can use **ws_06_02.sql** to do this.

```
SQL> @aton
SQL> get ws_06_02
 1 SELECT c.cust_last_name
 2 FROM customers c
 3 WHERE c.country_id = 52790
 4 AND c.cust_id NOT IN (
 5 SELECT /*+ &hint */ s.cust_id
 6* FROM sales s)
SQL> @ws_06_02
Enter value for hint: MERGE_AJ
```

**Student Observations:**

2. Drop all indexes on the `CUSTOMERS` table. Analyze the SQL statement in **`ws_06_03.sql`**.

```
SQL> @dai
on which table: customers
SQL> @atto
SQL> get ws_06_03
  1  SELECT country_name
  2  FROM   countries co
  3  WHERE  EXISTS (SELECT 'x'
  4                  FROM    customers c
  5                 WHERE  co.country_id = c.country_id
  6*                AND c.cust_credit_limit < 2000)
SQL> @ws_06_03
```

This is a `SELECT` statement with a subquery. Because the predicate with the subquery contains an `EXISTS` operator, this statement is known as a *semi-join*.

**Student Observations:**

Try to find the optimal performance of this query by providing hints such as `MERGE_SJ` and `HASH_SJ`. You can use **`ws_06_04.sql`** to provide the hints:

```
SQL> get ws_06_04
  1  SELECT country_name
  2  FROM   countries co
  3  WHERE  EXISTS (SELECT /*+ &hint */ 'x'
  4                  FROM    customers c
  5                 WHERE  co.country_id = c.country_id
  6*                AND c.cust_credit_limit < 2000)
SQL> @ws_06_04
Enter value for hint: MERGE_SJ
```

**Student Observations:**

Create an index on the `COUNTRY_ID` column in the `CUSTOMERS` table and retest your results.

```
SQL> @ci
on which table : customers
on which column(s): country_id
SQL> @ws_06_04
Enter value for hint: HASH_SJ
```

**Student Observations:**

3. Make sure to drop all indexes on the `SALES` table first and analyze the SQL statement in **`ws_06_05.sql`** using AUTOTRACE.

```
SQL> @dai
on which table: sales
```

```
SQL> get ws_06_05
 1 SELECT /*+ &hint */ count(*)
 2 FROM sales s
 3 WHERE exists (SELECT 'x'
 4 FROM customers c
 5* WHERE s.cust_id = c.cust_id)
SQL> @ws_06_05
Enter value for hint:HASH_SJ
```

This is also a semi-join statement (as in the previous exercise). Try providing hints to improve performance.

**Student Observations:**

4. Drop all existing indexes on the CUSTOMERS table and then create indexes on the following columns: CUST_GENDER, CUST_POSTAL_CODE, and CUST_CREDIT_LIMIT using the **ci.sql** script. Provide table and column names at the prompts. Check the indexes on the CUSTOMERS table using **li.sql**.

```
SQL> @dai
on which table: customers
SQL> @ci
on which table : customers
on which column(s): cust_gender
Creating index cust_cust_gender_idx …
SQL> @ci
on which table : customers
on which column(s): cust_postal_code
Creating index cust_cust_postal_code_idx …
SQL> @ci
on which table : customers
on which column(s): cust_credit_limit
Creating index cust_cust_credit_limit_idx …
SQL> @li
List indexes on table: customers
TABLE_NAME    INDEX_TYPE           INDEX_NAME
------------- -------------------- -------------------------
CUSTOMERS     UNIQUE               CUSTOMERS_PK
              NONUNIQUE            CUST_CUST_CREDIT_LIMIT_IDX
                                   CUST_CUST_GENDER_IDX
                                   CUST_CUST_POSTAL_CODE_IDX
```

5. Examine the **ws_06_06.sql** query. The statement contains an INDEX hint. Run this statement with different indexes and take notes about the results.

```
SQL> get ws_06_06
1 SELECT /*+ INDEX (c &Indexname) */
2 c.*
3 FROM customers c
4 WHERE cust_gender = 'M'
5 AND cust_postal_code = 40804
```

```
6 AND cust_credit_limit = 10000
SQL> @ws_06_06
Enter value for indexname: CUST_CUST_CREDIT_LIMIT_IDX
```

**Student Observations:**

```
SQL> @ ws_06_06
Enter value for indexname: CUST_CUST_GENDER_IDX
```

**Student Observations:**

```
SQL> @ws_06_06
Enter value for indexname: CUST_CUST_POSTAL_CODE_IDX
```

**Student Observations:**

6. Enable AUTOTRACE. Examine the **ws_06_07.sql** query. Now the statement contains an AND_EQUAL hint. This hint accepts two or more index names, forcing the optimizer to merge those indexes. Run this script with different index names and make notes of indexes being used. Run the index script to obtain index names. You can use **li.sql** to list indexes before you begin.

```
SQL> @li
List indexes on table :  customers
SQL> @atto
```

```
SQL> get ws_06_07
1 SELECT /*+ AND_EQUAL (c &index_name1, &index_name2) */
2 c.*
3 FROM customers c
4 WHERE cust_gender = 'M'
5 AND cust_postal_code = 40804
6* AND cust_credit_limit = 10000
SQL> @ws_06_07
Enter value for index_name: CUST_CUST_CREDIT_LIMIT_IDX
Enter value for index_name: CUST_CUST_GENDER_IDX
SQL> @atoff
```

Try more combinations for **index_name1** and **index_name2**. Also try to enter all three index names by entering two index names after one of the prompts.

**Student Observations:**

## Workshop 7: SQL Tuning Summary

### Advanced Workshop (Optional)

**Objectives:**

- Identify problematic SQL
- View execution plans
- Create indexes
- Restructure SQL statements

1. Open a terminal window. Change to workshop directory. Start SQL*Plus. Log in as **`SH`**/**`SH`**. To set up for this workshop, view and execute the script **`create_tab.sql`**. This workshop is designed for students to work in teams of two or more people.
2. **Scenario:** You have been receiving complaints that a specific application has been performing poorly. The SQL statements within that application have been captured in one script file called **`slow_application.sql`**. View the SQL statements and then use the tools discussed in the course to improve their performance. You may choose to create individual scripts for each statement or set of statements. Some files are provided for you ( **`stmt1.sql, stmt2.sql`**…).

   a. Identify the high-load statements. **Hint:** Look for  shared cursors, full table scans,  high disk reads, high values for parsed and elapsed times, and so on. You can use TKPROF, SQL*Plus AUTOTRACE or Top SQL to do this. **Hint:** If using SQLTrace and TKPROF, you may want to flush the shared pool first.

   b. After you have identified the bad SQL for each statement, try to come up with a way to get the optimizer to generate a better execution plan.

      i. Look at rewriting the SQL better.

      ii. Create indexes, histograms, materialized view, or use hints.

   c. Execute the new statement with any new access structures in place and verify that a new execution plan is generated and that it reduces the cost (CPU, elapsed) and is more efficient.

   **Note:**  The steps a, b, and c may need to be repeated for different combinations of indexes, hints, and so on, to determine the best solution.

   **Note:** There may be more than one correct solution.

```
SQL> get slow_application
 1  /* statement 1 */
 2  SELECT cust_first_name, cust_last_name, cust_credit_limit
 3  FROM test_customers
 4  WHERE cust_credit_limit =1500;
 5  SELECT cust_first_name, cust_last_name, cust_credit_limit
 6  FROM test_customers
 7  WHERE cust_credit_limit =3000;
```

```
SELECT cust_first_name, cust_last_name, cust_credit_limit
FROM test_customers
WHERE cust_credit_limit =11000;
SELECT cust_first_name, cust_last_name, cust_credit_limit
FROM test_customers
WHERE cust_credit_limit =5000;
SELECT cust_first_name, cust_last_name, cust_credit_limit
FROM test_customers
WHERE cust_credit_limit =1000;
SELECT cust_first_name, cust_last_name, cust_credit_limit
FROM test_customers
WHERE cust_credit_limit =9000;
SELECT cust_first_name, cust_last_name, cust_credit_limit
FROM test_customers
WHERE cust_credit_limit =10000;
SELECT cust_first_name, cust_last_name, cust_credit_limit
FROM test_customers
WHERE cust_credit_limit =15000;
SELECT cust_first_name, cust_last_name, cust_credit_limit
FROM test_customers
WHERE cust_credit_limit =11000;
SELECT cust_first_name, cust_last_name, cust_credit_limit
FROM test_customers
WHERE cust_credit_limit =7000;
/* statement 2 */
SELECT max(cust_credit_limit)
FROM test_customers
WHERE cust_city ='Paris';
/* statement 3 */
SELECT c.CUST_GENDER, c.CUST_MARITAL_STATUS,
sum(QUANTITY_SOLD),
sum(AMOUNT_SOLD)
FROM test_promotions p,test_sales s,test_customers c
WHERE p.promo_id=s.promo_id
  AND promo_name='NO PROMOTION #'
  AND s.cust_id=c.cust_id
GROUP BY c.CUST_GENDER, c.CUST_MARITAL_STATUS;
/* statement 4 */
 SELECT count(*)  FROM (
      SELECT PROMO_CATEGORY, sum(PROMO_COST)
      FROM test_promotions
      WHERE PROMO_NAME <> 'NO PROMOTION'
      GROUP BY PROMO_CATEGORY) ;
/* statement 5 */
SELECT  count(co.country_name)
      FROM test_countries co,test_customers cu
      WHERE co.country_id=cu.country_id
        AND cu.cust_credit_limit =
           (SELECT max(cust_credit_limit)
            FROM test_customers
            WHERE CUST_YEAR_OF_BIRTH = '1990')
        AND cu.CUST_YEAR_OF_BIRTH = '1990';
/* statement 6 */
 SELECT cust_id, SUM(quantity_sold)
```

```
      FROM sales
      WHERE prod_id = 30
      GROUP BY cust_id
      ORDER BY 2;
/*statement 7 */
SELECT cust_last_name, cust_first_name
FROM test_customers
WHERE upper(cust_last_name)like  'K%';
/* statement 8 */
SELECT cust_last_name, s.country_name
FROM test_customers c, test_countries s
WHERE c.country_id = s.country_id
AND substr(to_char(s.country_id), 1, 1) = '1';

```

You can use the SQL Tuning checklist provided to help you find solutions to these problematic queries.

**Note:** This workshop does not have a formal solution. Try to work in teams and then compare the results with other teams.

## SQL Tuning Checklist

1. Identify Statements to tune using:
    - ADDM
    - AWR
    - EM Top SQL
    - `V$SQL_AREA`
    - `V$SQL_TEXT`
    - Statspack
2. View execution statistics using:
    - ADDM
    - SQL*trace
    - `TKPROF`
    - `TRCCESS`
    - `DBMS_MONITOR`
    - Statspack
    - Make note of:
        - CPU time
        - Elapsed time
        - Disk reads
        - Disk sorts
3. Tune SQL automatically using SQL Tuning Advisor:
    - Optimizer stats analysis
    - SQL Profiling
    - Index analysis
    - SQL restructure

4. Tune SQL manually:

   a. Gather information about the underlying objects used in the SQL statements.

      - Obtain table, index, and column definitions.
      - Obtain view definitions.
      - Understand column data distribution.
        - Uniqueness
        - Nulls
        - Skew
      - Identify if data from more than one table is required by a statement resulting in joins.
      - Verify the join predicates to avoid Cartesian joins.
      - Verify presence of indexes.
      - Verify presence of Materialized views.
      - Verify type of indexes.

   b. Verify execution plans by using:

      - Explain plan
      - SQL*Plus `AUTOTRACE`

   c. Verify statistics in:

      - `User_tab_columns`
      - `User_indexes`
      - `User_tables`
      - Look for:
        - Last analyzed
        - Existence of histograms where appropriate

5. Verify that statistics are current:

   - Set Automatic statistics gathering to appropriate intervals.

- Back up existing statistics before gathering new ones.
- Use the `DBMS_STATS` package to gather statistics where statistics are stale or absent.
- Use Dynamic sampling on volatile objects if needed.

6. Change access paths.
   - Use SQL Access Advisor.
   - Use SQL Tuning Advisor.
   - Create  B*tree indexes on highly selective data.
   - Create bitmap indexes on low-cardinality columns.
   - Bitmap indexes help in queries using OR or aggregates.
   - Create bitmap join indexes to facilitate joins.
   - Create concatenated indexes to facilitate full index scans.
   - Create histograms on skewed data.
   - Create materialized views on queries involving joins and aggregates.
   - Keep in mind that:
     - Full table scans on small tables or queries retrieving a large percentage of rows are OK
     - A full index scan may be faster than a full table scan
     - An index skip scan may be faster than a full index scan
     - An index access by rowid may be faster than an index range scan
     - Use of distinct or GROUP BY may indicate a missing predicate
7. Restructure queries keeping the following in mind:
   a. Use SQL Tuning advisor.
   b. Inequality conditions cannot use indexes.
   c. Distinct causes sorts.
   d. GROUP BY causes sorts.
   e. Aggregates can use indexes.

f. Applying functions on indexed columns prevents the index from being used.

g. Low selectivity queries do not use indexes.

h. Use UNION ALL instead of UNION (wherever possible).

i. Nesting queries too deeply causes poor performance.

j. Use `EXISTS` instead of `IN` for subqueries to check for `TRUE` or `FALSE` values (wherever possible).

k. Use `NOT EXISTS` instead of `NOT IN` whenever possible.

l. Implicit or explicit conversions may cause an index not to be used.

m. OR and IN list conditions are not performance efficient.

n. If possible = or AND conditions are preferable.

8. Use hints to influence the optimizer in choosing:

   o. Query transformation

   p. Join orders

   q. Join methods

   r. Access paths

9. Verify whether the new code improves performance.

   - From a user perspective such as response time, time taken to run a report and so on

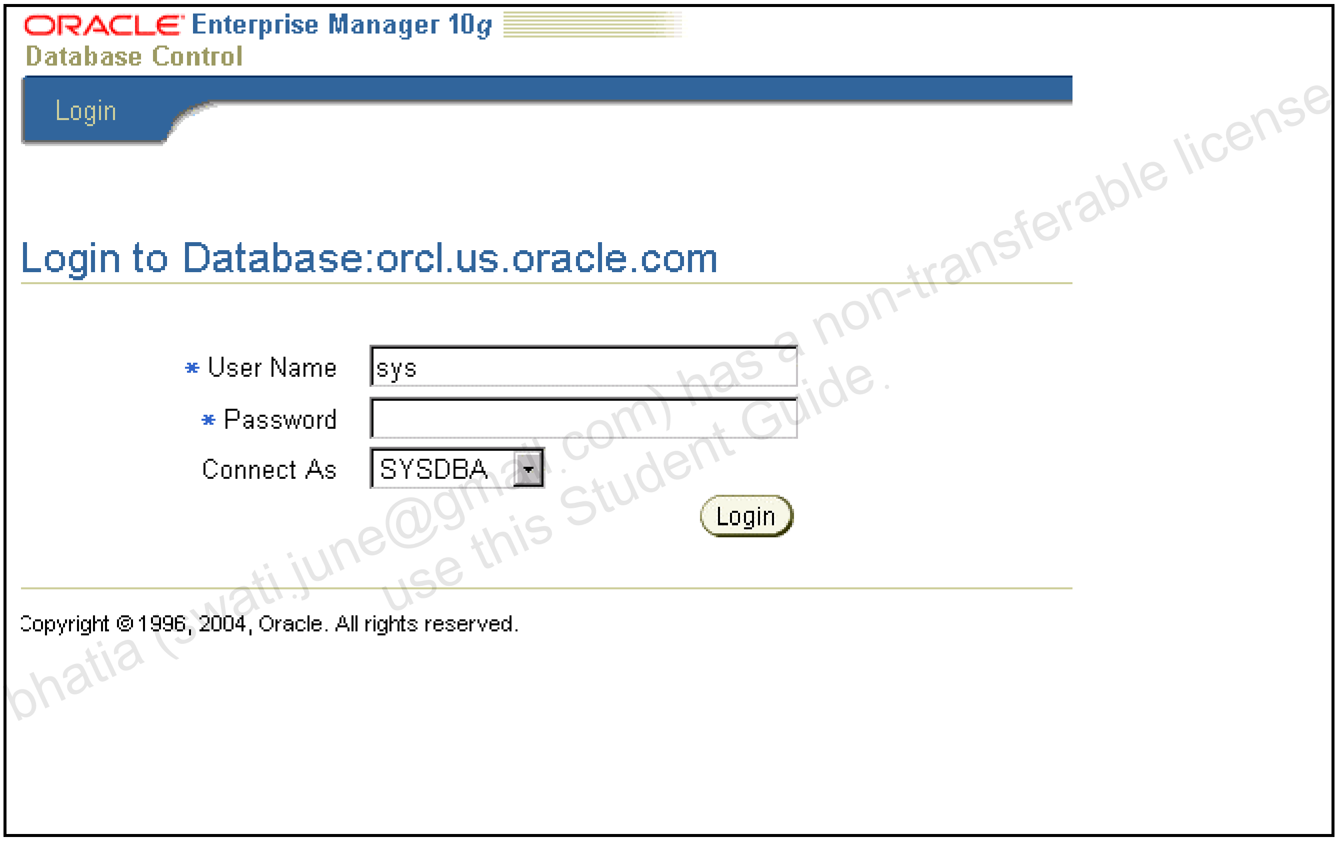
   - Check that the execution statistics (step 2) reflect the performance gain from the changes that you have made in CPU time, elapsed time, and so on, from a resource usage perspective.

# Appendix C: Practice Solutions

## Practice 1

1. Launch a browser and enter the URL `http:// machine_name:5500/em`, where *machine_name* stands for the IP address of your local machine. Log in to Enterprise Manager as `sys` with the password `SYSDBA`, and then click **Login**. If you are logging in for the first time, a license agreement may appear. Click **I Agree** to proceed.

   **Note:** You may also see an Adobe dialog box, in which you must accept an agreement before graphics can appear properly.



Login to Database:orcl.us.oracle.com

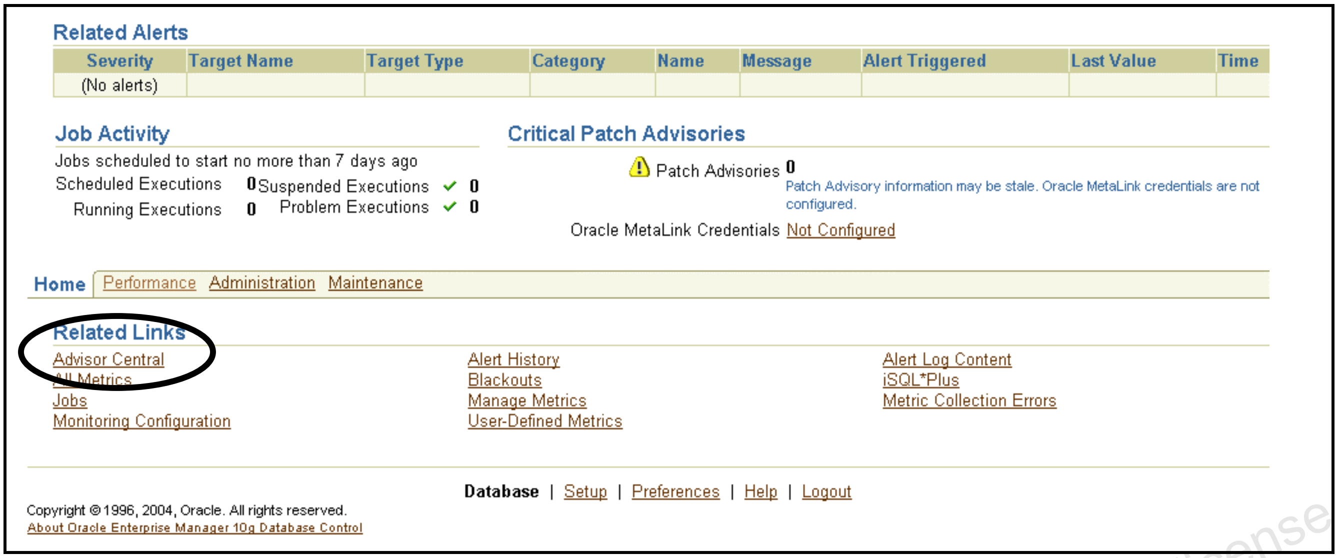
* User Name: sys

* Password:

Connect As: SYSDBA

Copyright © 1996, 2004, Oracle. All rights reserved.

2. Scroll to the bottom of the home page and click **Advisor Central** under **Related Links**.



**Related Alerts**

| Severity | Target Name | Target Type | Category | Name | Message | Alert Triggered | Last Value | Time |
|---|---|---|---|---|---|---|---|---|
| (No alerts) | | | | | | | | |

**Job Activity**

Jobs scheduled to start no more than 7 days ago

| | | | |
|---|---|---|---|
| Scheduled Executions | 0 | Suspended Executions ✓ | 0 |
| Running Executions | 0 | Problem Executions ✓ | 0 |

**Critical Patch Advisories**

⚠ Patch Advisories 0

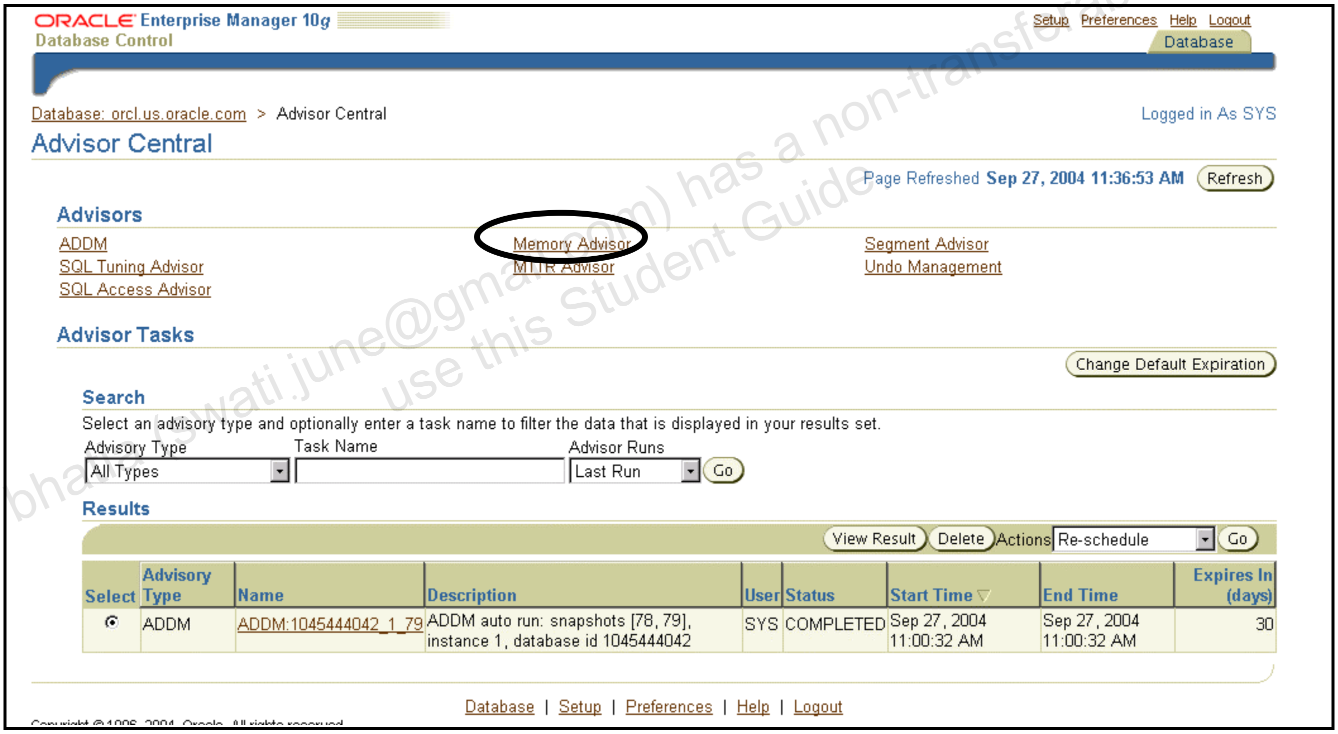
Patch Advisory information may be stale. Oracle MetaLink credentials are not configured.

Oracle MetaLink Credentials Not Configured

**Related Links**

| | | |
|---|---|---|
| Advisor Central | Alert History | Alert Log Content |
| All Metrics | Blackouts | iSQL*Plus |
| Jobs | Manage Metrics | Metric Collection Errors |
| Monitoring Configuration | User-Defined Metrics | |

3. Click the **Memory Advisor** link under **Advisors**.



ORACLE Enterprise Manager 10g
Database Control
Setup Preferences Help Logout
Database

Database: orcl.us.oracle.com > Advisor Central

Logged in As SYS

**Advisor Central**

Page Refreshed **Sep 27, 2004 11:36:53 AM** Refresh

**Advisors**

| | | |
|---|---|---|
| ADDM | Memory Advisor | Segment Advisor |
| SQL Tuning Advisor | MTTR Advisor | Undo Management |
| SQL Access Advisor | | |

**Advisor Tasks**

Change Default Expiration

**Search**

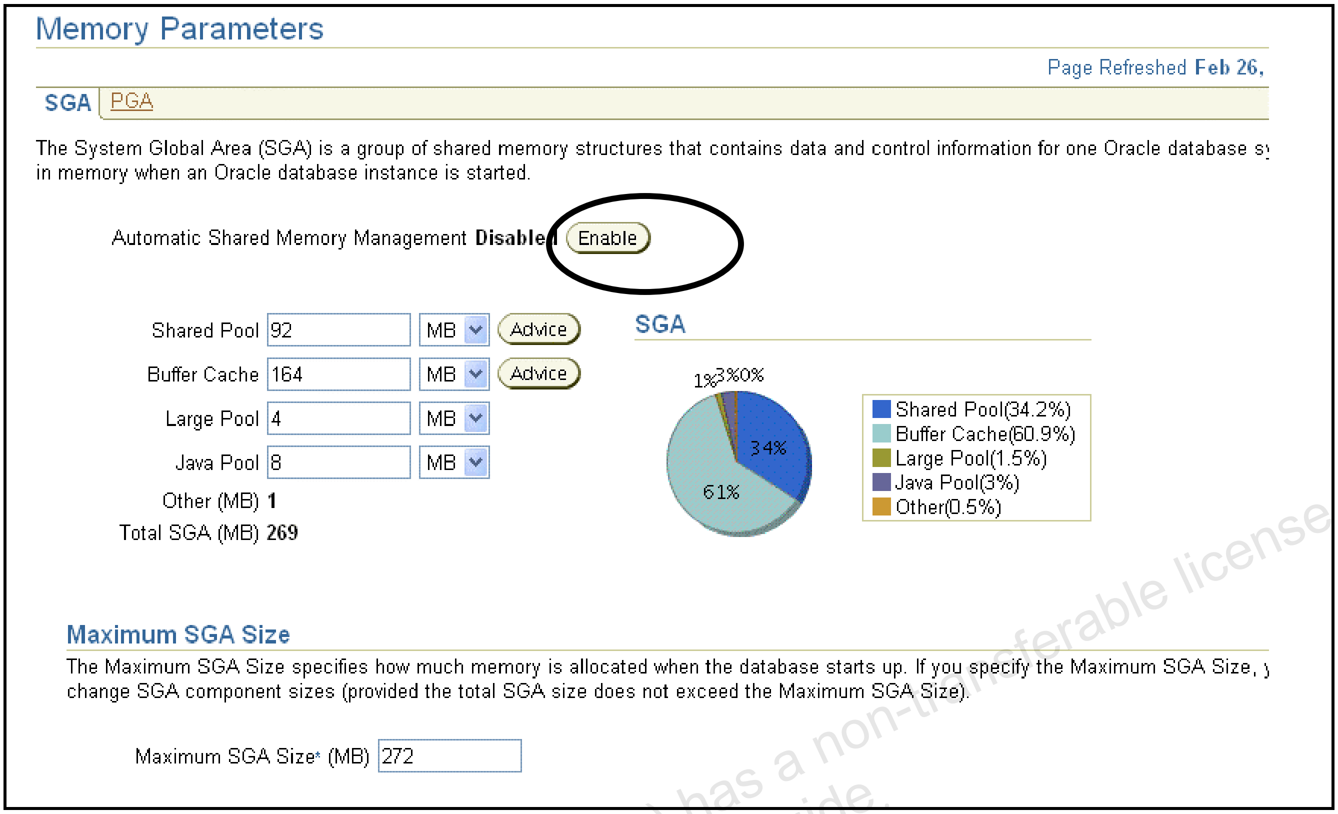
Select an advisory type and optionally enter a task name to filter the data that is displayed in your results set.

Advisory Type: All Types
Task Name:
Advisor Runs: Last Run
Go

**Results**

| Select | Advisory Type | Name | Description | User | Status | Start Time ▽ | End Time | Expires In (days) |
|---|---|---|---|---|---|---|---|---|
| ● | ADDM | ADDM:1045444042_1_79 | ADDM auto run: snapshots [78, 79], instance 1, database id 1045444042 | SYS | COMPLETED | Sep 27, 2004 11:00:32 AM | Sep 27, 2004 11:00:32 AM | 30 |

4. Click the **Enable** button for **Automatic Shared Memory Management**.



## Memory Parameters

Page Refreshed Feb 26,

SGA | PGA

The System Global Area (SGA) is a group of shared memory structures that contains data and control information for one Oracle database sy in memory when an Oracle database instance is started.

Automatic Shared Memory Management **Disabled** Enable

|  |  |  |  |
|---|---|---|---|
| Shared Pool | 92 | MB | Advice |
| Buffer Cache | 164 | MB | Advice |
| Large Pool | 4 | MB |  |
| Java Pool | 8 | MB |  |
| Other (MB) | **1** |  |  |
| Total SGA (MB) | **269** |  |  |

SGA

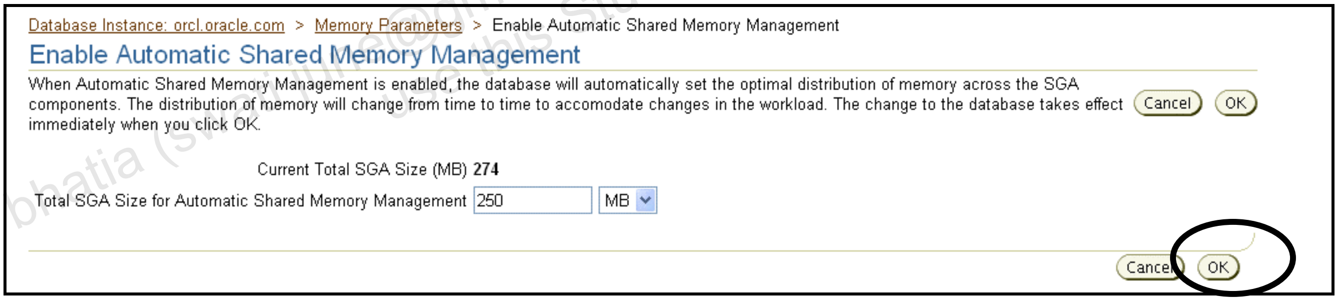
- 🟦 Shared Pool(34.2%)
- 🟦 Buffer Cache(60.9%)
- 🟩 Large Pool(1.5%)
- 🟪 Java Pool(3%)
- 🟧 Other(0.5%)

### Maximum SGA Size

The Maximum SGA Size specifies how much memory is allocated when the database starts up. If you specify the Maximum SGA Size, y change SGA component sizes (provided the total SGA size does not exceed the Maximum SGA Size).

Maximum SGA Size* (MB) 272

5. Set “Total SGA for Shared Memory Management” to 250 MB. Click **OK** to enable **Automatic Shared Memory Management**.



Database Instance: orcl.oracle.com > Memory Parameters > Enable Automatic Shared Memory Management

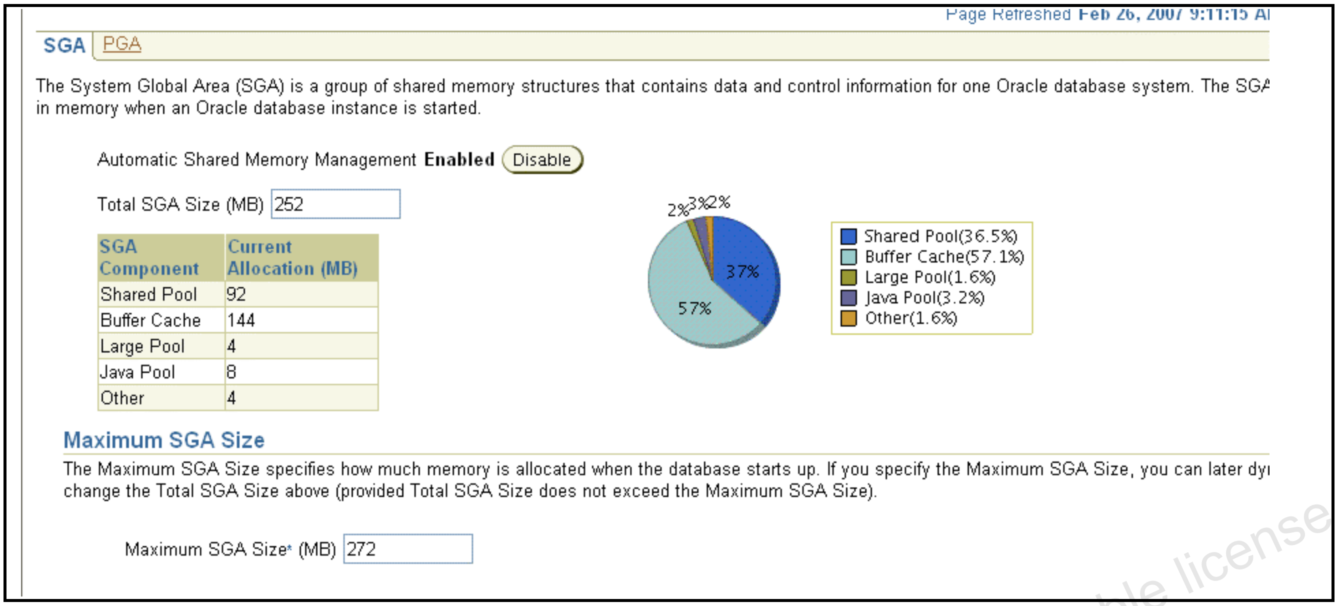
## Enable Automatic Shared Memory Management

When Automatic Shared Memory Management is enabled, the database will automatically set the optimal distribution of memory across the SGA components. The distribution of memory will change from time to time to accomodate changes in the workload. The change to the database takes effect immediately when you click OK.

Current Total SGA Size (MB) **274**

Total SGA Size for Automatic Shared Memory Management 250 MB

6. The Oracle database will now automatically adjust the settings for the various pools and caches according to the requirements of the workload.



Page Refreshed **Feb 26, 2007 9:11:15 AI**

**SGA** | PGA

The System Global Area (SGA) is a group of shared memory structures that contains data and control information for one Oracle database system. The SGA in memory when an Oracle database instance is started.

Automatic Shared Memory Management **Enabled** (Disable)

Total SGA Size (MB) 252

| SGA Component | Current Allocation (MB) |
|---|---|
| Shared Pool | 92 |
| Buffer Cache | 144 |
| Large Pool | 4 |
| Java Pool | 8 |
| Other | 4 |

**Maximum SGA Size**

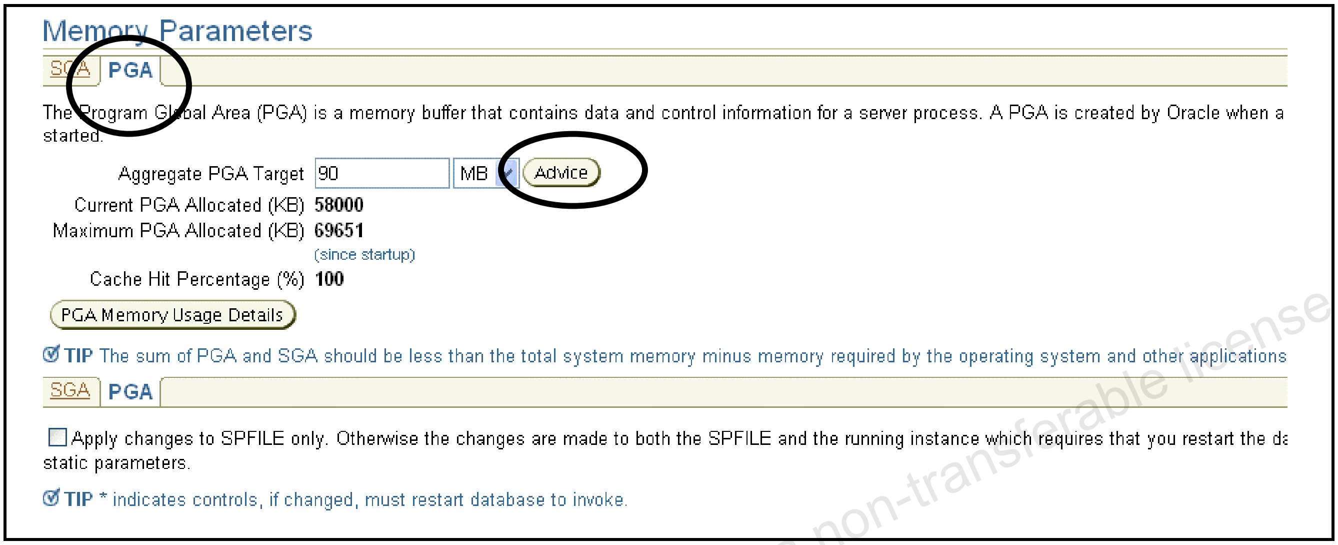
The Maximum SGA Size specifies how much memory is allocated when the database starts up. If you specify the Maximum SGA Size, you can later dy change the Total SGA Size above (provided Total SGA Size does not exceed the Maximum SGA Size).

Maximum SGA Size* (MB) 272

## Using the PGA Advisor

To allocate memory associated with the PGA, perform the following steps:

1. Click the **PGA** tab.



Memory Parameters

SGA | **PGA**

The Program Global Area (PGA) is a memory buffer that contains data and control information for a server process. A PGA is created by Oracle when a started.

Aggregate PGA Target 90 MB Advice

Current PGA Allocated (KB) **58000**

Maximum PGA Allocated (KB) **69651**
(since startup)

Cache Hit Percentage (%) **100**

PGA Memory Usage Details

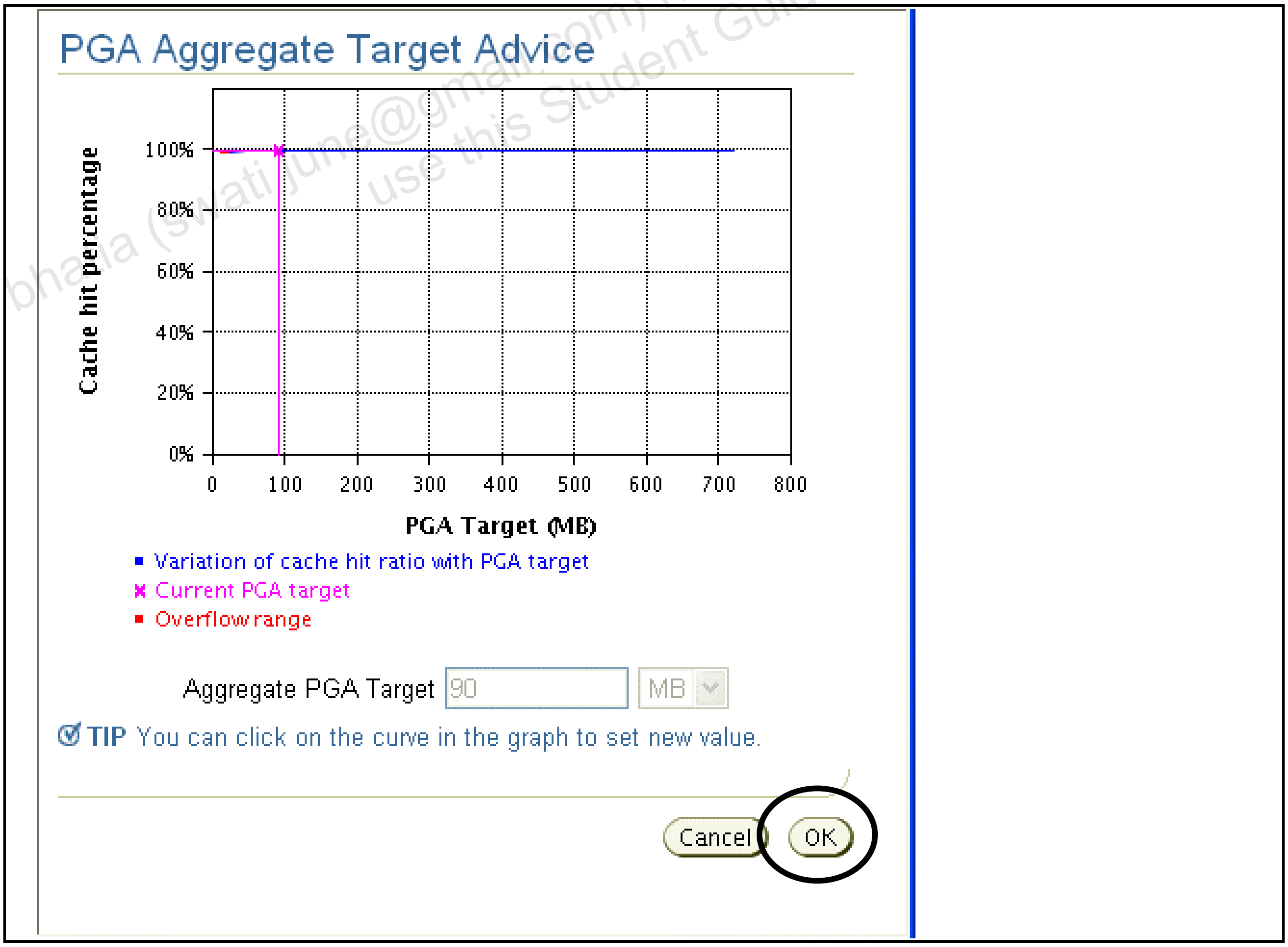
**TIP** The sum of PGA and SGA should be less than the total system memory minus memory required by the operating system and other applications

SGA | **PGA**

□ Apply changes to SPFILE only. Otherwise the changes are made to both the SPFILE and the running instance which requires that you restart the da static parameters.

**TIP** * indicates controls, if changed, must restart database to invoke.

2. Click the **Advice** button.



PGA Aggregate Target Advice

▪ Variation of cache hit ratio with PGA target
✖ Current PGA target
▪ Overflow range

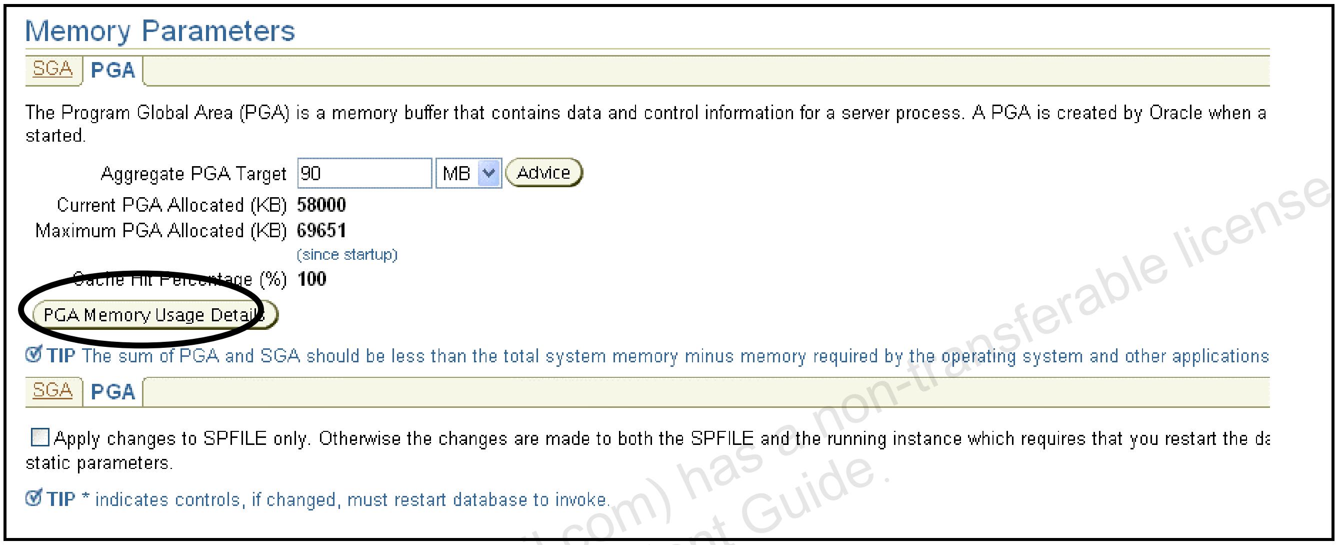
Aggregate PGA Target 90 MB

**TIP** You can click on the curve in the graph to set new value.

Cancel OK

3. The PGA Aggregate Target Advice graph shows the frequency with which data is found in the cache so that you do not have to access disk memory. In this case, note that the current PGA aggregate size is set to approximately 24 MB, and over 88% of all the requested services are obtained from memory. This also shows the overflow range, which starts around 12 MB. At 12 MB, the PGA requests use up the cache to around 90%. The PGA aggregate size implies that (based on current workloads and the number of sessions in the database) no more than 24 MB should be allocated for all PGA in this database. Click the **OK** button.

4. Click the **PGA Memory Usage Details** button.



Memory Parameters

The Program Global Area (PGA) is a memory buffer that contains data and control information for a server process. A PGA is created by Oracle when a started.

Aggregate PGA Target 90 MB

Current PGA Allocated (KB) 58000

Maximum PGA Allocated (KB) 69651 (since startup)

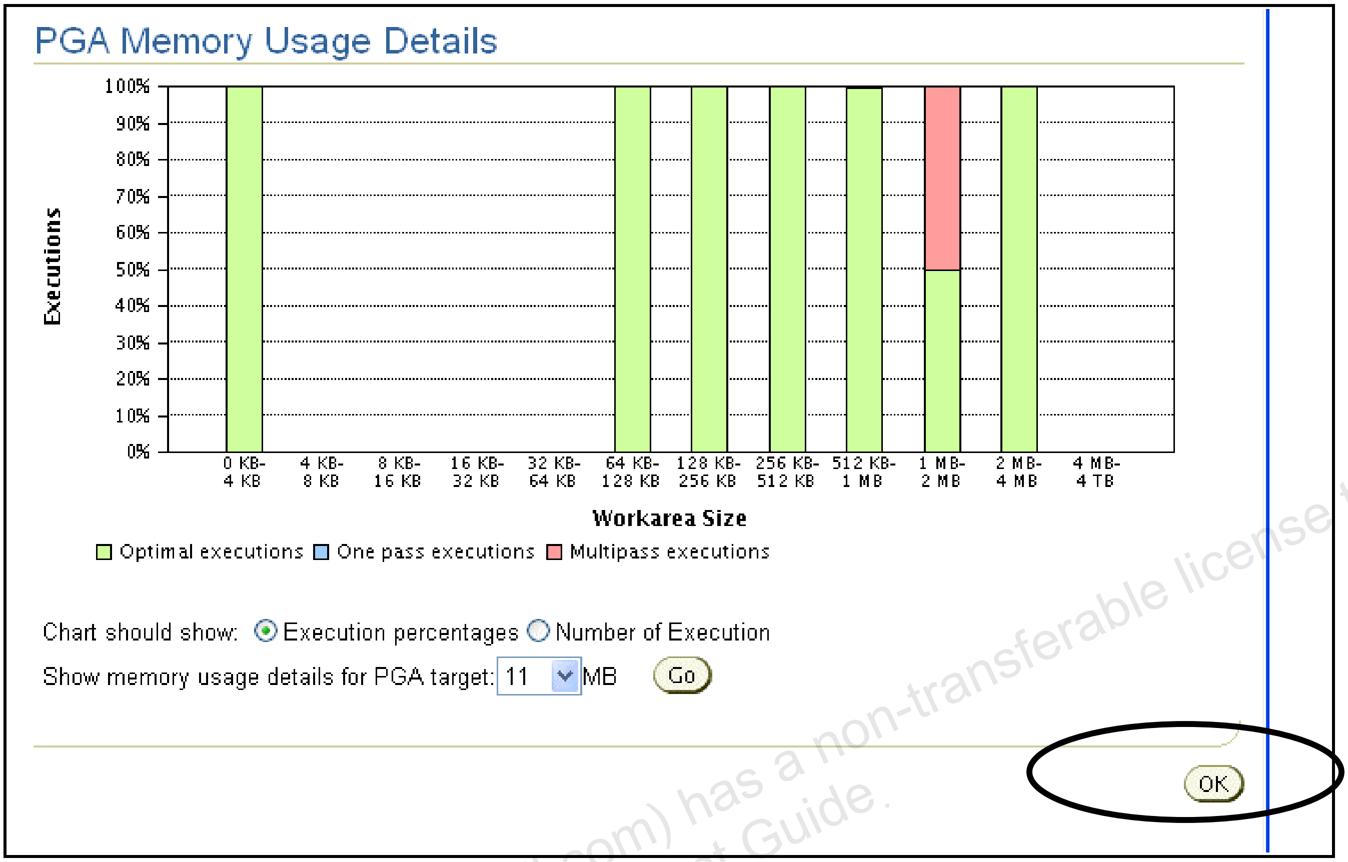
Cache Hit Percentage (%) 100

TIP The sum of PGA and SGA should be less than the total system memory minus memory required by the operating system and other applications

□ Apply changes to SPFILE only. Otherwise the changes are made to both the SPFILE and the running instance which requires that you restart the da static parameters.

TIP * indicates controls, if changed, must restart database to invoke.

5. This graph shows the usage details in memory size requests and execution percentages for various PGA memory requests. Click **OK** to proceed.



PGA Memory Usage Details

Chart should show: ● Execution percentages ○ Number of Execution

Show memory usage details for PGA target: 11 MB

## Practice 3

1. Open a terminal window. Set the current directory to **labs**. Log in to SQL*Plus as SH with password SH.

This lab demonstrates cursor sharing. Use **flush.sql** to flush the shared pool. View and run the scripts **lab_03_01.sql** and **lab_03_01b.sql**. These execute the same statement with different values in the predicate.

```
$ cd labs
$ sqlplus SH/SH
SQL> get flush
  1  --this script flushes the shared pool
  2  alter system flush shared_pool
  3* /
  4
SQL> @flush
System altered.
```

```
SQL> get lab_03_01
  1 SELECT * FROM customers WHERE cust_id = 10;
  2 SELECT * FROM customers WHERE cust_id =20;
  3 SELECT * FROM customers WHERE cust_id =50;
SQL> @lab_03_01
SQL> get lab_03_01b
  1  Variable cid number
  2  Exec :cid := 10
  3  SELECT * FROM customers WHERE cust_id =:cid;
  4  Exec :cid := 20
  5  SELECT * FROM customers WHERE cust_id =:cid;
  6  Exec :cid := 50
  7  SELECT * FROM customers WHERE cust_id =:cid;
SQL> @ lab_03_01b
```

2. Then use **lab_03_02.sql** to select from V$SQL_AREA. What do you see? Why?

```
SQL> get lab_03_02.sql
  1  SELECT sql_text, version_count, loads
  2  ,      invalidations, parse_calls, sorts
  3  FROM   v$sqlarea
  4  WHERE parsing_user_id > 0
  5   AND    command_type    = 3
  6   AND    lower(sql_text) like 'select * %my%'
  7*  ORDER  BY sql_text
SQL> @lab_03_02

SQL_TEXT       VERSION_COUNT    LOADS INVALIDATIONS PARSE_CALLS    SORTS
--------------- ------------ -------- ------------- -----------  --------
SELECT /* mysql             1        1             0           1        0
 */ * FROM cust
omers Where cus
t_id =10
```

```
SELECT /* mysql             1        1             0           1        0
 */ * FROM cust
omers Where cus
t_id =20

SELECT /* mysql             1        1             0           1        0
 */ * FROM cust
omers Where cus
t_id =50

SELECT  /* myne             1        1             0           3        0
wsql */ * FROM
customers WHERE
 cust_id =:cid

SELECT sql_text             1        1             0           1        1
, version_count
, loads ,
invalidations,
parse_calls, so
rts   FROM   v$
sqlarea   WHERE
 parsing_user_i
d > 0  AND    c
ommand_type
= 3  AND    low
er(sql_text) li
ke '%my%'  orde
r  by sql_text
```

You can see that the statement in **lab_03_01b.sql** is parsed once, and then the cursor is reused due to the usage of bind variables. The statement in **lab_03_01.sql**, however, is parsed every time for each new literal value.

## Practice 6

1. Open a terminal window. Set the current directory to **labs**. Start SQL*Plus. Log in to SH schema as SH with the password SH. Then check the existence and column structure of the CUSTOMERS table. View the structure of the PLAN_TABLE table.

   Also list the existing indexes on the CUSTOMERS table by using the **li.sql** script.

```
$ cd labs
$ sqlplus SH/SH
SQL> DESCRIBE customers
Name                                Null?    Type
 --------------------------------- -------- ----
 CUST_ID                            NOT NULL NUMbER
 CUST_FIRST_NAME                    NOT NULL VARCHAR2()
 CUST_LAST_NAME                     NOT NULL VARCHAR2()
 CUST_GENDER                        NOT NULL CHAR()
 CUST_YEAR_OF_BIRTH                 NOT NULL NUMBER
 CUST_MARITAL_STATUS                         VARCHAR2()
 CUST_STREET_ADDRESS                NOT NULL VARCHAR2()
 CUST_POSTAL_CODE                   NOT NULL VARCHAR2()
 CUST_CITY                          NOT NULL VARCHAR2()
…
SQL> DESCRIBE plan_table
Name                                   Null?    Type
 ------------------------------------- -------------
 STATEMENT_ID                                    VAR)
 PLAN_ID                                         NUMR
 TIMESTAMP                                       DATE
 REMARKS                                         VAR)
 OPERATION                                       VAR)
 OPTIONS                                         VAR)
 OBJECT_NODE                                     VAR)
 OBJECT_OWNER                                    VAR)
…

SQL> @li
List indexes on table: customers
TABLE_NAME                  INDEX_TYPE             INDEX_NAME
--------------------------- ---------------------- --------------------------
CUSTOMERS                   UNIQUE                 CUSTOMERS_PK
                            BITMAP                 CUSTOMERS_GENDER_BIX
                                                   CUSTOMERS_MARITAL_BIX
                                                   CUSTOMERS_YOB_BIX
```

Enable AUTOTRACE TRACEONLY EXPLAIN to suppress statement output and produce execution plans. Check the settings with SHOW AUTOTRACE.

```
SQL> show autotrace
autotrace OFF
SQL> set autotrace traceonly explain
SQL> show autotrace
autotrace TRACEONLY EXPLAIN
```

Disable **AUTOTRACE**.

```
SQL>set autotrace off
```

2. Explain the SQL statement that is provided in **`lab_06_02.sql`**.

```
SQL> get lab_06_02
  1  Explain plan for
  2  SELECT cust_first_name
  3  ,      cust_last_name
  4  FROM   customers
  5* WHERE  cust_id =100
SQL> @lab_06_02.sql
```

a. Use the **`rp.sql`** script to query the `PLAN_TABLE` table in a sophisticated manner:

```
SQL> get rp
  1  set feedback off
  2  SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
  3* set feedback on
SQL> @rp.sql
PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------
Plan hash value: 331174742

-------------------------------------------------------------------------------
| Id  | Operation                    | Name         | Rows  | Bytes | Cost %CPU|
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |              |     1 |    20 |     2  (0|
|   1 |  TABLE ACCESS BY INDEX ROWID| CUSTOMERS    |     1 |    20 |     2  (0|
|*  2 |   INDEX UNIQUE SCAN          | CUSTOMERS_PK |     1 |       |     1  (0|
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("CUST_ID"=100)
```

b. Enable SQL*Plus `AUTOTRACE` to display execution plans (**`attox.sql)`** and run the query again without the `EXPLAIN PLAN` command by using **`lab_06_02b.sql`**.

```
SQL> @attox
SQL> get lab_06_02b.sql
  1  SELECT cust_first_name, cust_last_name
  2  FROM   customers
  3* WHERE  cust_id = 100
SQL> @lab_06_02b.sql

Execution Plan
----------------------------------------------------------
0     SELECT STATEMENT Optimizer=ALL_ROWS (Cost=2 Card=1 Bytes=20)
1   0   TABLE ACCESS (BY INDEX ROWID) OF 'CUSTOMERS' (TABLE) (Cost=2 …)
2      1     INDEX (UNIQUE SCAN) OF 'CUSTOMERS_PK' (INDEX (UNIQUE)) (Cost=1 Card=1)
```

c. If you have some time left, enter your own SQL statements and experiment with `EXPLAIN  PLAN` and `AUTOTRACE`. Remember to disable `AUTOTRACE` when you are finished.

```
SQL>set autotrace off
```

3. Now retrieve information from `V$SQLAREA`. Execute the script **`lab_06_03.sql`**.

```
SQL> get lab_06_03
  1  Set termout off
  2  SELECT /* trialsql */ * FROM customers WHERE cust_id=10
  3  /
  4* Set termout on
  5
SQL> @lab_06_03
```

Obtain the **`sqlid`** for the statement that you executed. You can use **`sqlid.sql`** to help you do this.

```
SQL> get sqlid
  1  SELECT SQL_ID, SQL_TEXT FROM V$SQL
  2* WHERE SQL_TEXT LIKE '%trial%' ;
SQL> @sqlid

SQL_ID          SQL_TEXT
--------------  --------------------------------------------------------
…
8t0m2muj24pf5  SELECT /* trialsql */ * FROM customers WHERE cust_id=10
…
```

Using the `sqlid`, obtain the execution plan from the `V$SQLAREA` view. You can use the **`rpsqlarea.sql`** script to help do this. Remember to substitute your `sqlid` in the query.

```
SQL> SELECT PLAN_TABLE_OUTPUT FROM
     TABLE(DBMS_XPLAN.DISPLAY_CURSOR(' your sql id here'));

PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------
SQL_ID  8t0m2muj24pf5, child number 0
-------------------------------------
Select /* trialsql */ * FROM customers where cust_id = 10
Plan hash value: 331174742
-------------------------------------------------------------------------
| Id  | Operation                    | Name          | Rows  | Bytes
|Cost%CPU|
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |       |    |2 (100|
|   1 |  TABLE ACCESS BY INDEX ROWID| CUSTOMERS     |     1 |180 |2   (0|
|*  2 |   INDEX UNIQUE SCAN          | CUSTOMERS_PK |     1 |    |1   (0|
-------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("CUST_ID"=10)
```

You will see the execution plan for the statement.

4. Take a look at some SQL from the AWR.

a. Query V$SQL to obtain the sqlid for the statement containing the text "REPORT". You can run the script **lab_06_04a.sql** to execute this statement. First run **flush.sql** to flush the shared pool.

```
SQL>@flush.sql
SQL> get lab_06_04a
  1 set termout off
  2 SELECT /* REPORT */ cust_last_name, cust_first_name
  3 FROM customers
  4 WHERE cust_credit_limit = 5000
  5  set termout off
SQL>@lab_06_04a

SQL> SELECT SQL_ID, SQL_TEXT FROM V$SQL
WHERE SQL_TEXT LIKE '%REPORT%' ;

SQL_ID        SQL_TEXT
------------- -------------------------
…

b99knx7vq1xuc SELECT /* REPORT */ cust_
              last_name, cust_first_nam
              e FROM customers WHERE cu
              st_credit_limit = 5000

…
```

b. Using sqlid, verify that this statement has been captured in the DBA_HIST_SQLTEXT dictionary view.

```
SQL> SELECT SQL_ID, SQL_TEXT FROM dba_hist_sqltext
  2  WHERE SQL_ID = 'your sql id here';

no rows selected
```

c. If the query does not return rows, then it indicates that the statement has not yet been loaded in the AWR.You can take a manual AWR snapshot rather than wait for the next snapshot (which occurs every hour).

```

SQL> EXEC DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT('ALL');

PL/SQL procedure successfully completed.
```

Then check to see if it has been captured in DBA_HIST_SQLTEXT using the same query as the previous step.

```
SQL> SELECT SQL_ID, SQL_TEXT FROM dba_hist_sqltext
  2  WHERE SQL_ID = 'your sql id here'

SQL_ID        SQL_TEXT
------------- -------------------------
b99knx7vq1xuc SELECT /* REPORT */ cust_
              last_name, cust_first_nam
```

```
               e FROM customers WHERE
               cust_credit_limit = 5000


1 row selected.
```

d. Use the **DBMS_XPLAN.DISPLAY_AWR ()** function to retrieve the execution plan.

```
SQL>SELECT PLAN_TABLE_OUTPUT
FROM
TABLE (DBMS_XPLAN.DISPLAY_AWR('your sql id here'));


PLAN_TABLE_OUTPUT
-----------------------------------------------------------------
SQL_ID b99knx7vq1xuc
--------------------
SELECT /* REPORT */ cust_last_name, cust_first_name FROM customers
WHERE cust_credit_limit = 5000

Plan hash value: 3624098912

-------------------------------------------------------------------------
| Id  | Operation         | Name      | Rows  | Bytes | Cost (%CPU)| Time
|
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |           |       |       |   335 (100)|
|
|   1 |  TABLE ACCESS FULL| CUSTOMERS |  6938 |   128K|   334   (3) |…
-------------------------------------------------------------------------
14 rows selected.
```

## Practice 7

1. Open a terminal window. Set the current directory to **`labs`**. Start SQL*Plus. Log in to `SH` schema as `SH` with the password `SH`. Verify the existence of the statistics-gathering job by querying from `DBA_SCHEDULER_JOBS`.

```
SQL> SELECT owner, job_name,enabled
  1  FROM DBA_SCHEDULER_JOBS
  2  WHERE JOB_NAME = 'GATHER_STATS_JOB';

OWNER                          JOB_NAME                       ENABL

------------------------------ ------------------------------ -----
SYS                            GATHER_STATS_JOB               TRUE

1 row selected.
```

2. Create a `MY_CUST` table that is identical to the `CUSTOMERS` table. You can use **`lab_07_02.sql`** to do this if you like.

```
SQL> CREATE table my_cust AS SELECT * FROM customers;
```

List the indexes on the table using the **`li.sql`** script.

```
SQL> @li
List indexes on table : my_cust
```

3. Query `USER_TABLES` to verify the existence of statistics. You can use the **`tabstats.sql`** script to do this.

```
SQL> get tabstats.sql
  1  SELECT last_analyzed analyzed, sample_size,monitoring,
  2  table_name
  3  FROM user_tables
  4* WHERE table_name = upper('&table_name')
 SQL> @tabstats
 on which table : my_cust

ANALYZED  SAMPLE_SIZE MON TABLE_NAME
--------- ----------- --- -------------------------
                      YES MY_CUST


1 row selected.
```

4. Run the query in **`lab_07_04.sql`** on the table, and then view the execution plan using **`attox.sql`**. This script sets `AUTOTRACE` to `TRACE ONLY EXPLAIN`.

```
SQL> @attox
SQL> get lab_07_04
  1* SELECT * FROM my_cust WHERE cust_id < 50
SQL> @lab_07_04
Execution Plan
----------------------------------------------------------
Plan hash value: 2970555845
```

```
-----------------------------------------------------------------------------
| Id  | Operation         | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |          |    49 |  8820 |   330   (1)| 00:00:04 |
|*  1 |  TABLE ACCESS FULL| MY_CUST  |    49 |  8820 |   330   (1)| 00:00:04 |
-----------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("CUST_ID"<50)
```

5. Disable `sqltrace` by using **`atoff.sql`**. Create an index on the `CUST_ID` column by using the **`ci.sql`** script.

```
SQL>@atoff
SQL> @ci
     on which table    : my_cust
     on which column(s): cust_id

creating index new_cust_id_idx ...
```

6. Run the query again and view the execution plan using `sqltrace`.

```
SQL> @attox
SQL> get lab_07_04
  1* SELECT * FROM my_cust WHERE cust_id < 50
SQL> @lab_07_04
Execution Plan
----------------------------------------------------------
Plan hash value: 2257762371

--------------------------------------------------------------------------------
| Id  | Operation                   | Name              | Rows  | Bytes | Cost (|
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |                   |    49 |  8820 |    51 |
|   1 |  TABLE ACCESS BY INDEX ROWID| MY_CUST           |    49 |  8820 |    51 |
|*  2 |   INDEX RANGE SCAN          | MY_C_CUST_ID_IDX  |    49 |       |     2 |
-------------------------------------------------------------------------

Predicate Information (identified by operation id):

 2 - access("CUST_ID"<50)
```

7. How does the optimizer know to use the index? Disable `AUTOTRACE` first by using **`atoff.sql`**.

   **Hint:** Query `USER_INDEXES` and `USER_TABLES` by using **`tabstats.sql`** and **`indstats.sql`** to get the answer.

```
SQL> @atoff
SQL> get tabstats.sql
  1  SELECT last_analyzed analyzed, sample_size,monitoring,
  2  table_name
  3  FROM user_tables
  4  WHERE table_name = upper('&table_name')
  5 undef table_name
 SQL> @tabstats
 On which table: my_cust
ANALYZED  SAMPLE_SIZE MON TABLE_NAME
--------- ----------- --- -------------------------
                      YES MY_CUST

1 row selected.
```

```
SQL > get indstats.sql
  1  SELECT index_name name, num_rows n_r,
  2  last_analyzed l_a, distinct_keys d_k,leaf_blocks l_b,
  3  avg_leaf_blocks_per_key a_l,join_index j_I
  4  FROM user_indexes
  5* WHERE table_name = upper('&table_name')
SQL> @indstats
on which table : my_cust

NAME                            N_R      L_A         D_K      L_B      A_L I
------------------------------ -------- ---------    --------  --------- --------
MY_C_CUST_ID_IDX           55500  19-OCT-04    55500        123     10

1 row selected.
```

8. Drop the index that you created. Then verify the index statistics again. What do you see?

```
SQL> @dai
on which table : my_cust

SQL> get indstats.sql
  1  SELECT index_name name, num_rows n_r,
  2  last_analyzed l_a, distinct_keys d_k,
  3  leaf_blocks l_b, avg_leaf_blocks_per_key
  4  a_l,join_index j_I
  5  FROM user_indexes
  6 WHERE table_name = upper('&table_name')
SQL>@indstats.sql
on which table : my_cust

no rows selected
```

9. Identify the date of the last analysis, and sample size for all tables in your schema. You can use the **schemastats.sql** script to help you do this.

```

SQL> get schemastats.sql
   1  SELECT last_analyzed analyzed,
   2      sample_size, monitoring, table_name
   3* FROM user_tables
SQL>@schemastats.sql

ANALYZED  SAMPLE_SIZE MON TABLE_NAME
--------- ----------- --- --------------------
26-FEB-07         503 YES PROMOTIONS
26-FEB-07       55500 YES CUSTOMERS
```

```
26-FEB-07          23 YES COUNTRIES
26-FEB-07        4500 YES SUPPLEMENTARY_DEMOGR
                          APHICS

26-FEB-07           0 YES MVIEW$_EXCEPTIONS
26-FEB-07          48 YES CAL_MONTH_SALES_MV
26-FEB-07       11266 YES FWEEK_PSCAT_SALES_MV
26-FEB-07        1826 YES TIMES
26-FEB-07          72 YES PRODUCTS
26-FEB-07           5 YES CHANNELS
 …
26-FEB-07       32286 YES SALES
26-FEB-07           0 YES COSTS
…
22-FEB-05           0 NO  SALES_TRANSACTIONS_E
                          XT
```

Identify the types of histograms for all columns in your schema. Try running the **stats.sql** script. Do you see any difference?

**Hint:** Query USER_TAB_COL_STATISTICS or use the **colhist.sql** script.

```

SQL> get colhist.sql
  1 SELECT column_name, num_distinct, num_buckets,histogram
  2 FROM USER_TAB_COL_STATISTICS
  3 WHERE histogram <> 'NONE'
SQL> @colhist


COLUMN_NAME                    NUM_DISTINCT NUM_BUCKETS HISTOGRAM
------------------------------ ------------ ----------- ---------------
SID                                      16          16 FREQUENCY
SERIAL#                                   5           5 FREQUENCY

2 rows selected.

SQL>get stats.sql
 1  EXECUTE
 2   DBMS_STATS.GATHER_SCHEMA_STATS('SH',DBMS_STATS.AUTO_SAMPLE_SIZE);
SQL>@stats.sql

PL/SQL procedure successfully completed.
SQL>@colhist
COLUMN_NAME                    NUM_DISTINCT NUM_BUCKETS HISTOGRAM
------------------------------ ------------ ----------- ---------------
CUST_CREDIT_LIMIT                         8           8 FREQUENCY
CUST_ID                               55500         254 HEIGHT BALANCED
ORD_TOTAL                             10000         200 HEIGHT BALANCED
CUST_ID                                   3           3 FREQUENCY

4 rows selected.
```

10. Now consider bind-variable peeking. First run **flush.sql** to flush the shared pool. Then run the script **lab_07_10.sql** that creates the NEW_CUST table and populates it with skewed data. It also creates an index and a histogram on the skew data column CUST_ID. After this script is run, the CUST_ID column has 1,000 rows with a value of 1, one row with a value of 2, and one row with a value of 3.

```
SQL> get flush
 1   ALTER SYSTEM FLUSH SHARED_POOL;
SQL> /
SQL> get lab_07_10.sql
  1  column operations format a20
  2  column object_name format a20
  3  column options format a20
  4  create table new_cust(cust_id number, ord_total number)
  5  /
  6  declare
  7  i integer;
  8  begin
  9    for i in 1..10000 loop
 10      insert into new_cust values(1, i);
 11    end loop;
 12    insert into new_cust values(2, 10000);
 13    insert into new_cust values(3, 1500);
 14    commit;
 15  end;
 16  /
 17  --Create an index on cust_id
 18  create index new_cust_cust_id on new_cust(cust_id);
 19  --Create histogram with 200 buckets
 20  begin
 21  dbms_stats.gather_table_stats(ownname => 'SH', tabname =>
'new_cust',
 22  method_opt => 'FOR ALL COLUMNS SIZE 200');
 23  end;
 24* /

SQL> @lab_07_10
```

Now run **lab_07_10a.sql** and use **rp.sql** to get the execution plan.

```
SQL> get lab_07_10a
  1  explain plan for
  2  SELECT count(ord_total) FROM new_cust
  3* where cust_id = 1;
SQL> @lab_07_10a
SQL> @rp
PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------
Plan hash value: 2621169891
-------------------------------------------------------------------------
| Id  | Operation          | Name | Rows   | Bytes | Cost (%CPU)| Time
|
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |         |   1 |     6    | 6  (0)| 00:00:01 |
|    1 |   SORT AGGREGATE   |          |    1 |     6    |       |          |
|*   2 |    TABLE ACCESS FULL| NEW_CUST | 10000 | 60000 | 6 (0)| 00:00:01 |
-------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - filter("CUST_ID"=1)
```

You see that this is a full table scan. Now try running **lab_07_10b.sql**. Is the index used?

```
SQL> get lab_07_10b
  1  explain plan for
  2  SELECT count(ord_total) FROM new_cust
  3* WHERE cust_id = 2
SQL> @lab_07_10b
SQL> @rp
PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------
Plan hash value: 543115811
-------------------------------------------------------------------------
| Id  | Operation                | Name               | Rows  | Bytes | Cost |
-------------------------------------------------------------------------
-----
|   0 | SELECT STATEMENT             |                   |     1 |     6 |
2|
|   1 |  SORT AGGREGATE             |                   |     1 |     6 |      |
|   2 |   TABLE ACCESS BY INDEX ROWID| NEW_CUST          |     1 |     6 |     2|
|*  3 |    INDEX RANGE SCAN         | NEW_CUST_CUST_ID |     1 |       |     1|
------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("CUST_ID"=2)
```

The optimizer uses the histogram to determine whether to use an index.

## Practice 8

1. Open a terminal window. Set the current directory to **`labs`**. Start SQL*Plus. Log in to SQL*Plus as `SH` with password `SH`. Make sure that `AUTOTRACE` is disabled by using **`atoff.sql`**.

```
SQL> @atoff
```

2. Provide an identifier for the trace file to help you locate it. Drop all indexes on the **`CUSTOMERS`** table. Enable SQL Trace. Analyze the SQL statement in **`lab_08_01.sql`** by using SQL Trace and TKPROF.

```
SQL> ALTER SESSION SET TRACEFILE_IDENTIFIER = 'User12';

SQL> @dai
On which table : customers

SQL> ALTER SESSION SET SQL_TRACE = TRUE;

SQL> get lab_08_01
  1  SELECT max(cust_credit_limit)
  2  FROM customers
  3  WHERE cust_city ='Paris';
SQL>@lab_08_01

MAX(CUST_CREDIT_LIMIT)
----------------------
                 15000

1 row selected.
```

3. Now create an index on the `CUST_CITY` column by using **`ci.sql`**, and then run the same query again.

```
SQL> @ci
on which table     : customers
on which column(s): cust_city

SQL> get lab_08_01
  1  SELECT max(cust_credit_limit)
  2  FROM customers
  3  WHERE cust_city ='Paris';
SQL>@lab_08_01

MAX(CUST_CREDIT_LIMIT)
----------------------
                 15000

1 row selected.
```

4. Disable tracing.

```
SQL> ALTER SESSION SET SQL_TRACE = false;
```

5. Determine the location of the trace files by using the SHOW PARAMETER DUMP command and making note of the UDUMP destination.

```
SQL> SHOW PARAMETER dump
name                       TYPE        value
-------------------------- ----------- --------------------
background_core_dump       string      partial
background_dump_dest       string      /u01/app/oracle/admin/orcl/cdump
core_dump_dest             string      /u01/app/oracle/admin/orcl/bdump
max_dump_file_size         string      UNLIMITED
shadow_core_dump           string      partial
user_dump_dest             string      /u01/app/oracle/admin/orcl/udump
```

6. Exit your session.

```
SQL>exit
```

7. Change directory to the **UDUMP** destination.

```
$ cd /u01/app/oracle/admin/orcl/udump
```

8. Locate your file by the file identifier that you gave at step 2. Look for a file called **orcl_ora_xxxxx_filename** (for example, orcl_ora_123456_User12.trc).

```
$ more <your file name>.trc
$ tkprof <your file name>.trc output.txt
```

9. View the difference in execution plans and statistics of the SQL statement with and without an index. You can use **gedit** to do this. Change back to your home directory and then to the **labs** directory.

```

$ gedit output.txt
$  cd
$  cd  labs
```

10. Take a look at DBMS_MONITOR. Start two sessions, one connected as SYS/ORACLE as SYSDBA and the other connected as SH.

11. From the SYSDBA session, determine the session ID (**sid**) and serial number (**serial#**) from v$session for the SH user, and then describe the DBMS_MONITOR package. Then, from the SYSDBA session, enable tracing using the sid and serial# values for the other session, including the waits and bind information, with the **lab_08_10.sql** script:

```
SQL> get  lab_08_10.sql
  1  DESCRIBE dbms_monitor
  2  SELECT sid, serial#
  3  FROM   v$session
  4  WHERE  username = 'SH';
  5
  6  EXEC DBMS_MONITOR.SESSION_TRACE_ENABLE ( -
  7  session_id => &sid ,        -
  8  serial_num => &serial ,     -
  9  waits      => true ,        -
 10 binds      => true ) ;
```

```
SQL> @lab_08_10
PROCEDURE CLIENT_ID_STAT_DISABLE
 Argument Name                  Type                    In/Out
Default?
------------------------------ ----------------------- ------ --------
 CLIENT_ID                      VARCHAR2                IN
PROCEDURE CLIENT_ID_STAT_ENABLE
 Argument Name                  Type                    In/Out
Default?
------------------------------ ----------------------- ------ --------
 CLIENT_ID                      VARCHAR2                IN
PROCEDURE CLIENT_ID_TRACE_DISABLE
 Argument Name                  Type                    In/Out
Default?
------------------------------ ----------------------- ------ --------
 CLIENT_ID                      VARCHAR2                IN
PROCEDURE CLIENT_ID_TRACE_ENABLE
 Argument Name                  Type                    In/Out
Default?
------------------------------ ----------------------- ------ --------
 CLIENT_ID                      VARCHAR2                IN
 WAITS                          BOOLEAN                 IN     DEFAULT
 BINDS                          BOOLEAN                 IN     DEFAULT
PROCEDURE SERV_MOD_ACT_STAT_DISABLE
 Argument Name                  Type                    In/Out
Default?
------------------------------ ----------------------- ------ --------
 SERVICE_NAME                   VARCHAR2                IN
 MODULE_NAME                    VARCHAR2                IN
 ACTION_NAME                    VARCHAR2                IN     DEFAULT
PROCEDURE SERV_MOD_ACT_STAT_ENABLE
 Argument Name                  Type                    In/Out
Default?
------------------------------ ----------------------- ------ --------
 SERVICE_NAME                   VARCHAR2                IN
 MODULE_NAME                    VARCHAR2                IN
 ACTION_NAME                    VARCHAR2                IN     DEFAULT
PROCEDURE SERV_MOD_ACT_TRACE_DISABLE
 Argument Name                  Type                    In/Out
Default?
------------------------------ ----------------------- ------ --------
 SERVICE_NAME                   VARCHAR2                IN
 MODULE_NAME                    VARCHAR2                IN     DEFAULT
 ACTION_NAME                    VARCHAR2                IN     DEFAULT
 INSTANCE_NAME                  VARCHAR2                IN     DEFAULT
PROCEDURE SERV_MOD_ACT_TRACE_ENABLE
 Argument Name                  Type                    In/Out
Default?
------------------------------ ----------------------- ------ --------
 SERVICE_NAME                   VARCHAR2                IN
 MODULE_NAME                    VARCHAR2                IN     DEFAULT
 ACTION_NAME                    VARCHAR2                IN     DEFAULT
 WAITS                          BOOLEAN                 IN     DEFAULT
 BINDS                          BOOLEAN                 IN     DEFAULT
 INSTANCE_NAME                  VARCHAR2                IN     DEFAULT
PROCEDURE SESSION_TRACE_DISABLE
 Argument Name                  Type                    In/Out
Default?
------------------------------ ----------------------- ------ --------
 SESSION_ID                     BINARY_INTEGER          IN     DEFAULT
 SERIAL_NUM                     BINARY_INTEGER          IN     DEFAULT
PROCEDURE SESSION_TRACE_ENABLE
 Argument Name                  Type                    In/Out
Default?
------------------------------ ----------------------- ------ --------
 SESSION_ID                     BINARY_INTEGER          IN     DEFAULT
 SERIAL_NUM                     BINARY_INTEGER          IN     DEFAULT
 WAITS                          BOOLEAN                 IN     DEFAULT
 BINDS                          BOOLEAN                 IN     DEFAULT



     SID  SERIAL#
-------- --------
     236    10327

Enter value for sid: 236
Enter value for serial: 10327

PL/SQL procedure successfully completed.
```

12. From the SH session, execute the **lab_08_11.sql** script, and then exit your session.

```
SQL> get lab_08_11.sql
  1  SELECT c.cust_last_name,  t.calendar_year,
  2       sum(s.amount_sold)
  3       FROM   sales     s JOIN
  4              customers c USING (cust_id) JOIN
  5              times t USING (time_id)
  6       GROUP BY c.cust_last_name, t.calendar_year
  7*      ORDER BY c.cust_last_name, t.calendar_year
SQL> @ lab_08_11.sql


…
Zimmerman                                        1998        37338.23
Zimmerman                                        1999        32201.97
Zimmerman                                        2000        58806.81
Zimmerman                                        2001        71710.31
Zoldos                                           1998        46096.77
Zoldos                                           1999        71375.51
Zoldos                                           2000        12383.68
Zoldos                                           2001        84018.39
Zwolinsky                                        1998        11475.14
Zwolinsky                                        2000         3817.61
```

```
3026 rows selected.
```

13. From the remaining SYSDBA session, determine your USER_DUMP_DEST location, locate the trace file, and view the contents. Determine the location of the trace files by using the SHOW PARAMETER DUMP command and making note of the UDUMP destination.

```
SQL> SHOW PARAMETER dump
name                          TYPE        value
-------------------------- ----------- --------------------
background_core_dump          string      partial
background_dump_dest          string      /u01/app/oracle/admin/orcl/cdump
core_dump_dest                string      /u01/app/oracle/admin/orcl/bdump
max_dump_file_size            string      UNLIMITED
shadow_core_dump              string      partial
user_dump_dest                string      /u01/app/oracle/admin/orcl/udump
```

14. Exit your session.

```
SQL>exit
```

15. Change directory to the UDUMP destination that you retrieved by the previous query.

```
$ cd /u01/app/oracle/admin/orcl/udump
```

16. Locate your file with the following command:

```
$ls -lt
```

17. View the top file that is the most recent:

```
$ more <your file name.trc>
```

18. Convert the file to readable format using the following command:

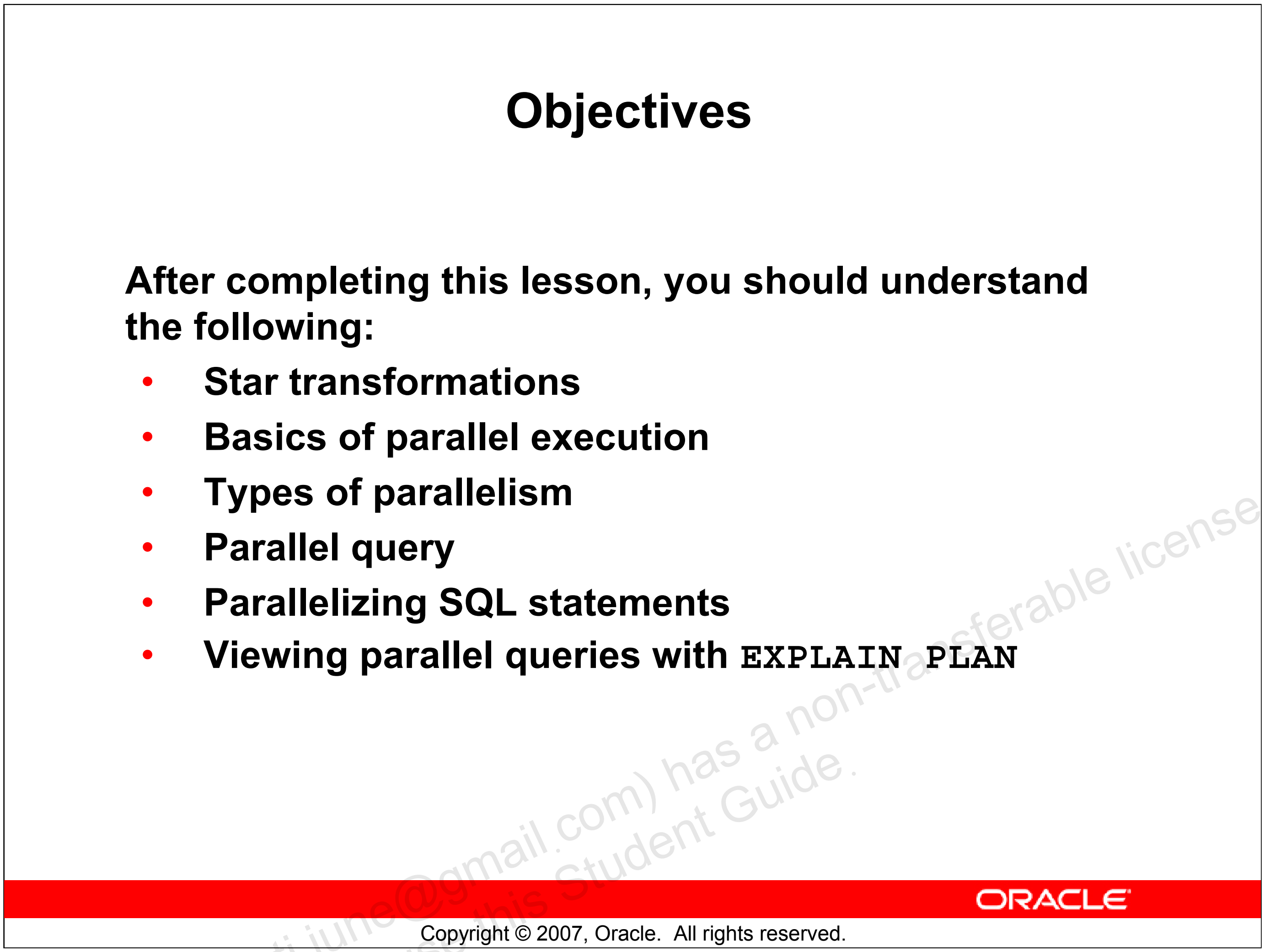
```
$ tkprof <your file name.trc> monitor.txt
```

19. You can use gedit to view the file. Change back to your home directory and then to the **labs** directory.

```
$ gedit monitor.txt
$ cd
$ cd labs
```

D

# Data Warehouse Tuning Considerations



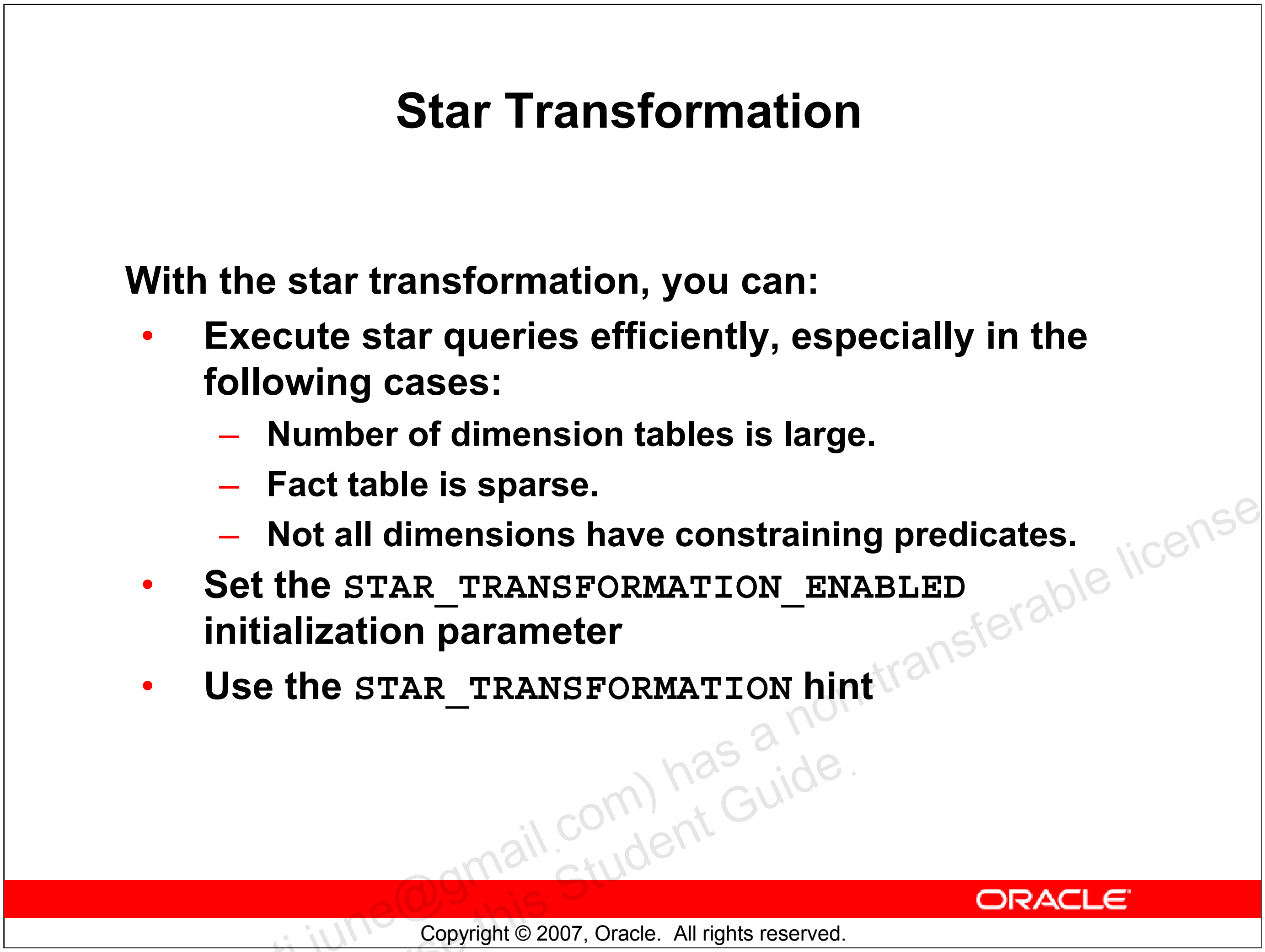
# Objectives

After completing this lesson, you should understand the following:

- Star transformations
- Basics of parallel execution
- Types of parallelism
- Parallel query
- Parallelizing SQL statements
- Viewing parallel queries with `EXPLAIN PLAN`

## Objectives

In this lesson, you learn when to use parallel execution. You also learn about the situations in which parallel execution is most beneficial to performance



## Star Transformation

With the star transformation, you can:

- Execute star queries efficiently, especially in the following cases:
  - Number of dimension tables is large.
  - Fact table is sparse.
  - Not all dimensions have constraining predicates.
- Set the `STAR_TRANSFORMATION_ENABLED` initialization parameter
- Use the `STAR_TRANSFORMATION` hint

### Star Transformation

The star transformation is a query transformation that helps execute star queries efficiently. The star transformation is especially effective if any of the following conditions exist:

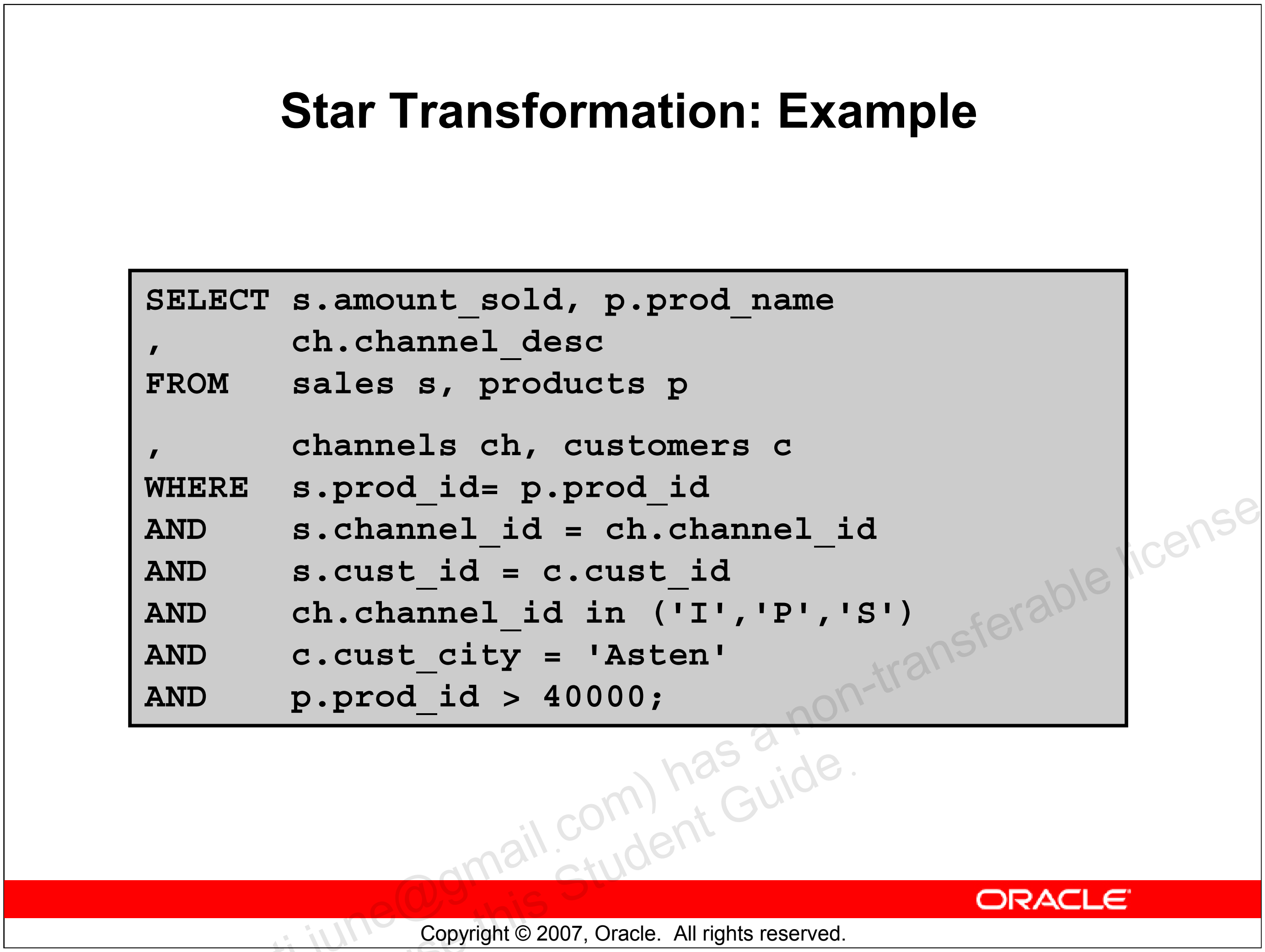
- The number of dimension tables is large, making the Cartesian product of the dimension tables too expensive.
- The fact table is sparse.
- There are queries in which not all dimension tables have constraining predicates (no additional nonjoin predicates).

The transformation can decide to combine bitmap indexes corresponding precisely to the constrained dimensions and does not need to produce the Cartesian product of the dimension tables, which is the basis of the other approach to tuning star queries.

The `STAR_TRANSFORMATION_ENABLED` parameter is dynamic and can be set to `TRUE` at the session level with the `ALTER SESSION` command:

```
    ALTER SESSION SET star_transformation_enabled = true;
```

Alternatively, you can specify the `STAR_TRANSFORMATION` hint at the statement level.



# Star Transformation: Example

```
SELECT s.amount_sold, p.prod_name
,      ch.channel_desc
FROM   sales s, products p
,      channels ch, customers c
WHERE  s.prod_id= p.prod_id
AND    s.channel_id = ch.channel_id
AND    s.cust_id = c.cust_id
AND    ch.channel_id in ('I','P','S')
AND    c.cust_city = 'Asten'
AND    p.prod_id > 40000;
```

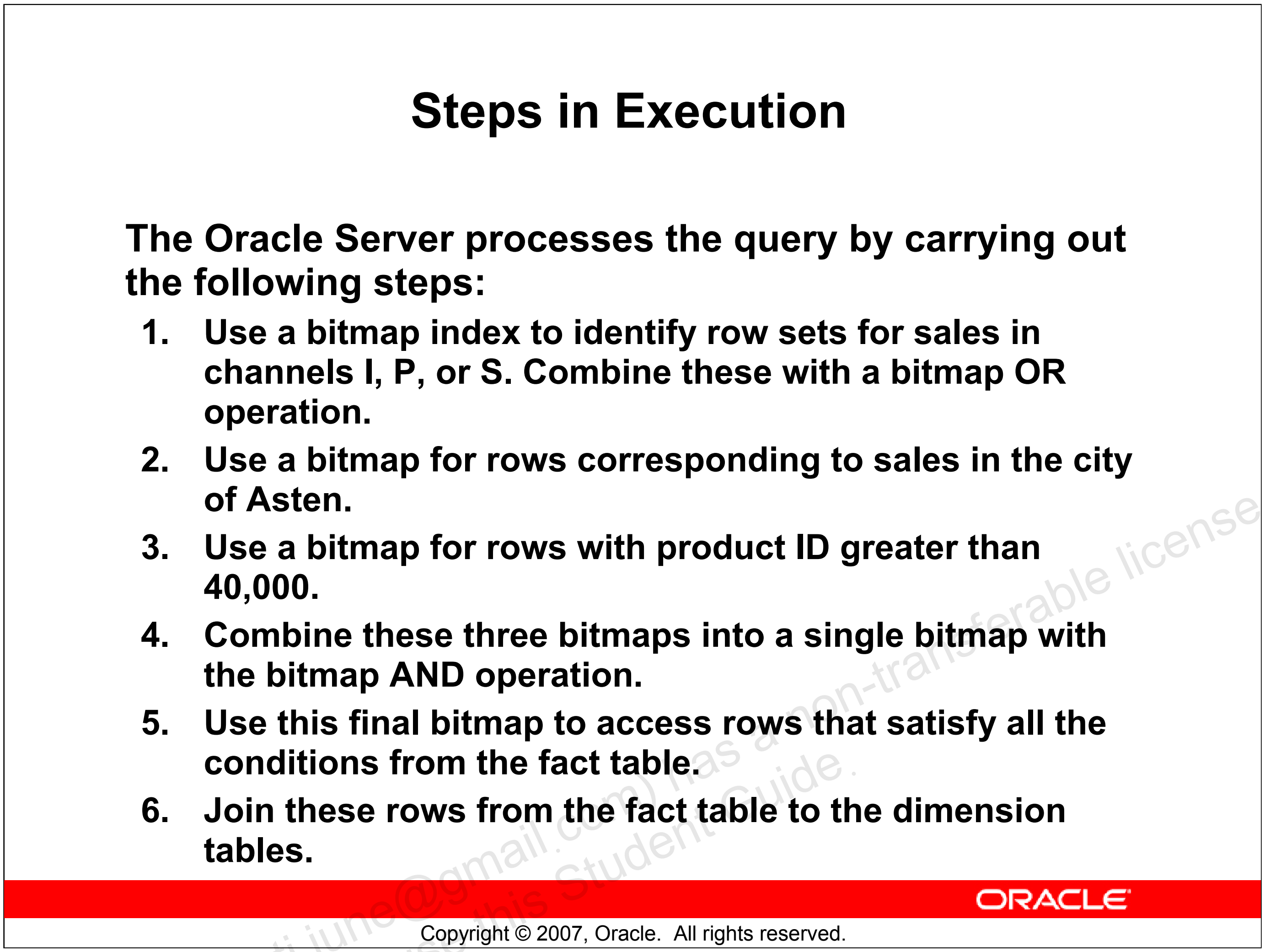
## Star Transformation: Example

In the example in the slide, the SALES table is the fact table, and CHANNELS, PRODUCTS, and CUSTOMERS are the dimension tables.

Because there are bitmap indexes on the CHANNEL_ID, PROD_ID, and CUST_ID columns of the SALES table, the query can be rewritten so that these bitmap indexes can be used.

Note that you are joining four tables but selecting only column values of three of them; the CUSTOMERS table is used only to select customers that are in the city of Asten. Note also that there are no nonjoin predicates on the facts table.

**Note:** It is not possible to force star transformation, and there are several constraints. The optimizer frequently does not consider a star transformation because a better execution plan is available or because one of the constraints for the star transformation is violated. For more information about the star transformation, see the *Oracle Database 10g: Data Warehousing Guide.*



## Steps in Execution

The Oracle Server processes the query by carrying out the following steps:

1. Use a bitmap index to identify row sets for sales in channels I, P, or S. Combine these with a bitmap OR operation.
2. Use a bitmap for rows corresponding to sales in the city of Asten.
3. Use a bitmap for rows with product ID greater than 40,000.
4. Combine these three bitmaps into a single bitmap with the bitmap AND operation.
5. Use this final bitmap to access rows that satisfy all the conditions from the fact table.
6. Join these rows from the fact table to the dimension tables.

### Steps in Execution

The Oracle Server processes this query in two phases. In the first phase, the Oracle Server uses the bitmap indexes on the foreign key columns of the fact table to identify and retrieve only the necessary rows from the fact table. In this star query, the bitmap index on `channel_id` is used to identify the three sets of all rows in the fact table corresponding to each of the channels I, P, and S. These sets are combined with a bitmap `OR` operation Additional set operations are done for the `customer` dimension and the `product` dimension. At this point in the star query processing, there are three bitmaps. Each bitmap corresponds to a separate dimension table, and each bitmap represents the set of rows in the fact table that satisfy that individual dimension's constraints. These three bitmaps are combined into a single bitmap using the bitmap `AND` operation.

This final bitmap represents the set of rows in the fact table that satisfy all of the constraints on the dimension table. This is the *result set*, which is the exact set of rows from the fact table that is needed to evaluate the query. Note that none of the actual data in the fact table has been accessed. All of these operations rely solely on the bitmap indexes and the dimension tables. Because of the bitmap indexes' compressed data representations, the bitmap set–based operations are extremely efficient.

### Steps in Execution (continued)

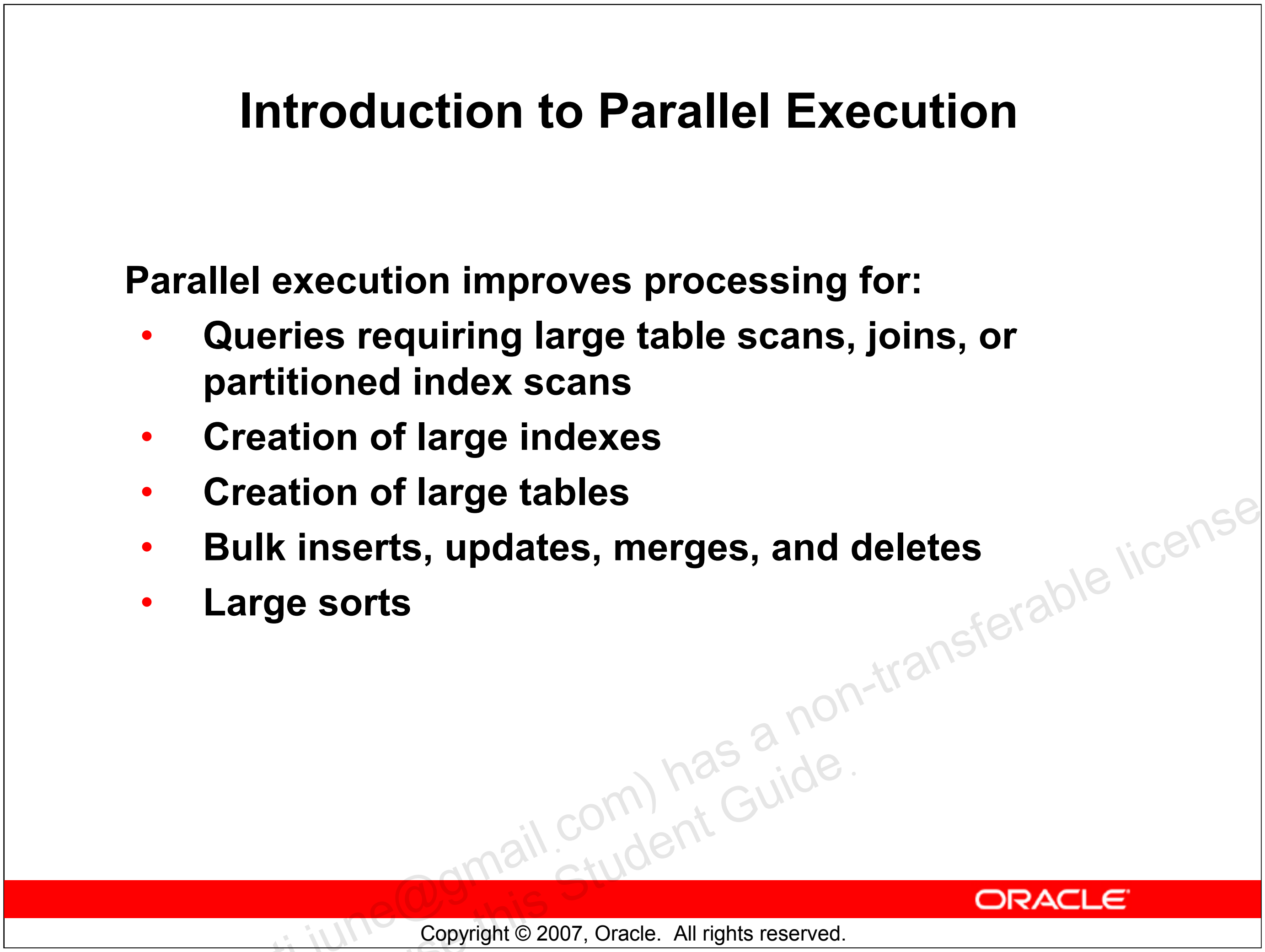
After the result set is identified, the bitmap is used to access the actual data from the sales table. Only those rows that are required for the end user's query are retrieved from the fact table. At this point, the Oracle Server has effectively joined all of the dimension tables to the fact table using bitmap indexes.

This technique provides excellent performance because the Oracle Server is joining all of the dimension tables to the fact table with one logical join operation, rather than joining each dimension table to the fact table independently.

The second phase of this query is to join these rows from the fact table (the result set) to the dimension tables. The Oracle Server uses the most efficient method for accessing and joining the dimension tables. The optimizer automatically determines which access method is most appropriate for a given dimension table, based upon the optimizer's knowledge about the sizes and data distributions of each dimension table.

The complete execution plan for the query is as follows:

```
Execution Plan
----------------------------------------------------------
   0    SELECT STATEMENT Optimizer=CHOOSE (Cost=2079 Card=1173
        Bytes=97359)
   1    2     RECURSIVE EXECUTION OF 'SYS_LE_2_0'
   2    0   TEMP TABLE TRANSFORMATION
   3    2     HASH JOIN (Cost=2079 Card=1173 Bytes=97359)
   4    3       TABLE ACCESS (FULL) OF 'SYS_TEMP_0FD9D660A_3A5D053'
                (Cost=2 Card=1568 Bytes=62720)
   5    3       HASH JOIN (Cost=2075 Card=3755 Bytes=161465)
   6    5         INLIST ITERATOR
   7    6           TABLE ACCESS (BY INDEX ROWID) OF 'CHANNELS'
                    (Cost=2 Card=3 Bytes=36)
   8    7             INDEX (RANGE SCAN) OF 'CHAN_PK' (UNIQUE)
                      (Cost=1 Card=1)
   9    5         HASH JOIN (Cost=2072 Card=6259 Bytes=194029)
  10    9           TABLE ACCESS (FULL) OF 'CUSTOMERS' (Cost=106
                     Card=81 Bytes=1215)
  11    9           PARTITION RANGE (ALL)
  12   11             TABLE ACCESS (BY LOCAL INDEX ROWID) OF 'SALES'
                     (Cost=1963 Card=121769 Bytes=1948304)
  13   12               BITMAP CONVERSION (TO ROWIDS)
  14   13                 BITMAP AND
  15   14                   BITMAP OR
  16   15                     BITMAP INDEX (SINGLE VALUE) OF
                                'SALES_CH_BIX'
  17   15                     BITMAP INDEX (SINGLE VALUE) OF
                                'SALES_CH_BIX'
  18   15                     BITMAP INDEX (SINGLE VALUE) OF
                                'SALES_CH_BIX'
  19   14                   BITMAP MERGE
  20   19                     BITMAP INDEX (RANGE SCAN) OF
                                'SALES_PROD_BIX'
```



# Introduction to Parallel Execution

**Parallel execution improves processing for:**

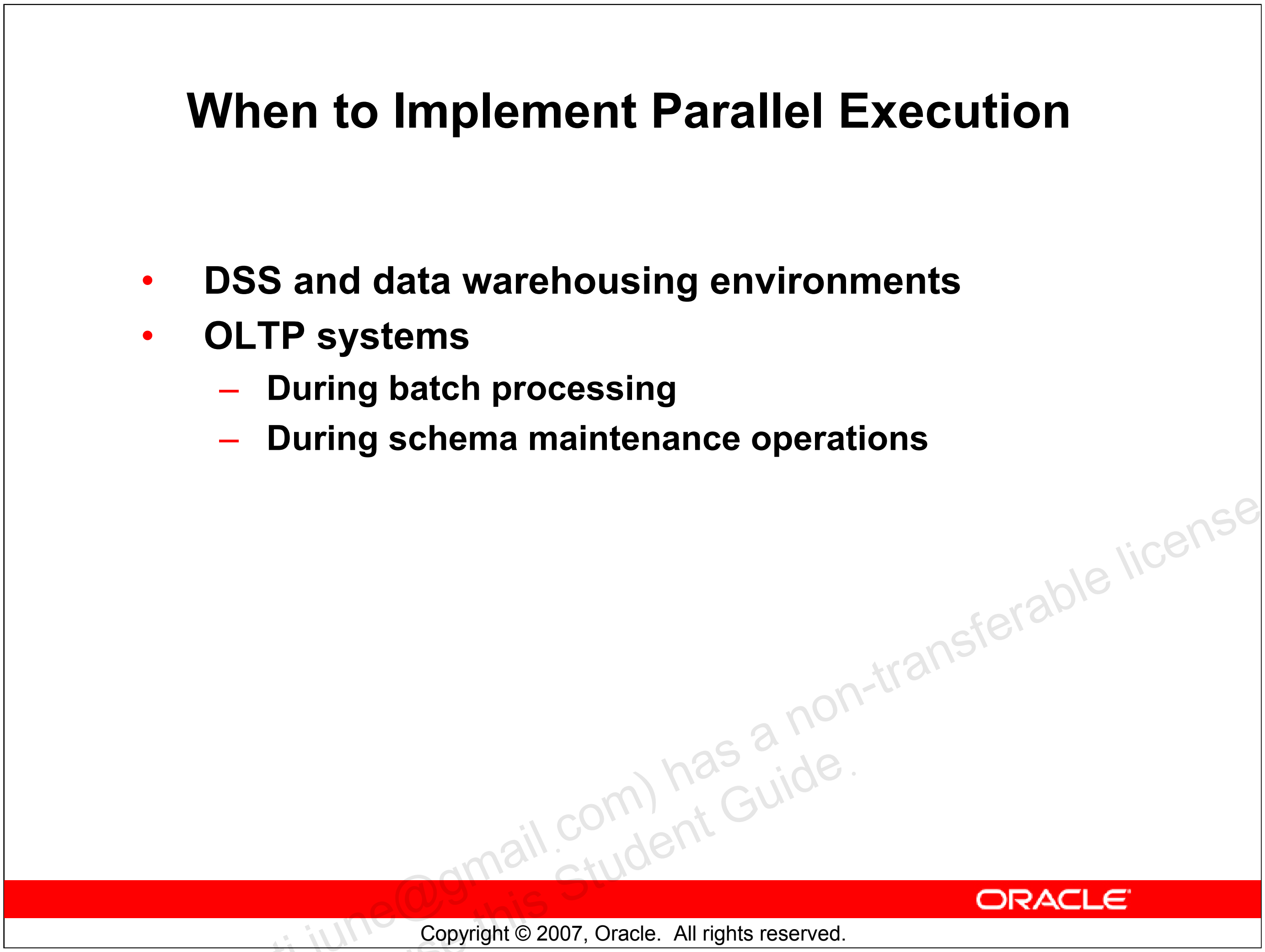
- Queries requiring large table scans, joins, or partitioned index scans
- Creation of large indexes
- Creation of large tables
- Bulk inserts, updates, merges, and deletes
- Large sorts

## Introduction to Parallel Execution

Parallel execution dramatically reduces response time for data-intensive operations on large databases that are typically associated with decision support systems (DSS) and data warehouses. You can also implement parallel execution on certain types of online transaction processing (OLTP) and hybrid systems. Parallel execution improves processing for:

- Queries requiring large table scans, joins, or partitioned index scans
- Creation of large indexes
- Creation of large tables (including materialized views)
- Bulk inserts, updates, merges, and deletes
- Sorting large volumes of data

You can also use parallel execution to access object types within an Oracle database. For example, you can use parallel execution to access large objects (LOBs).



## When to Implement Parallel Execution

- DSS and data warehousing environments
- OLTP systems
  - During batch processing
  - During schema maintenance operations

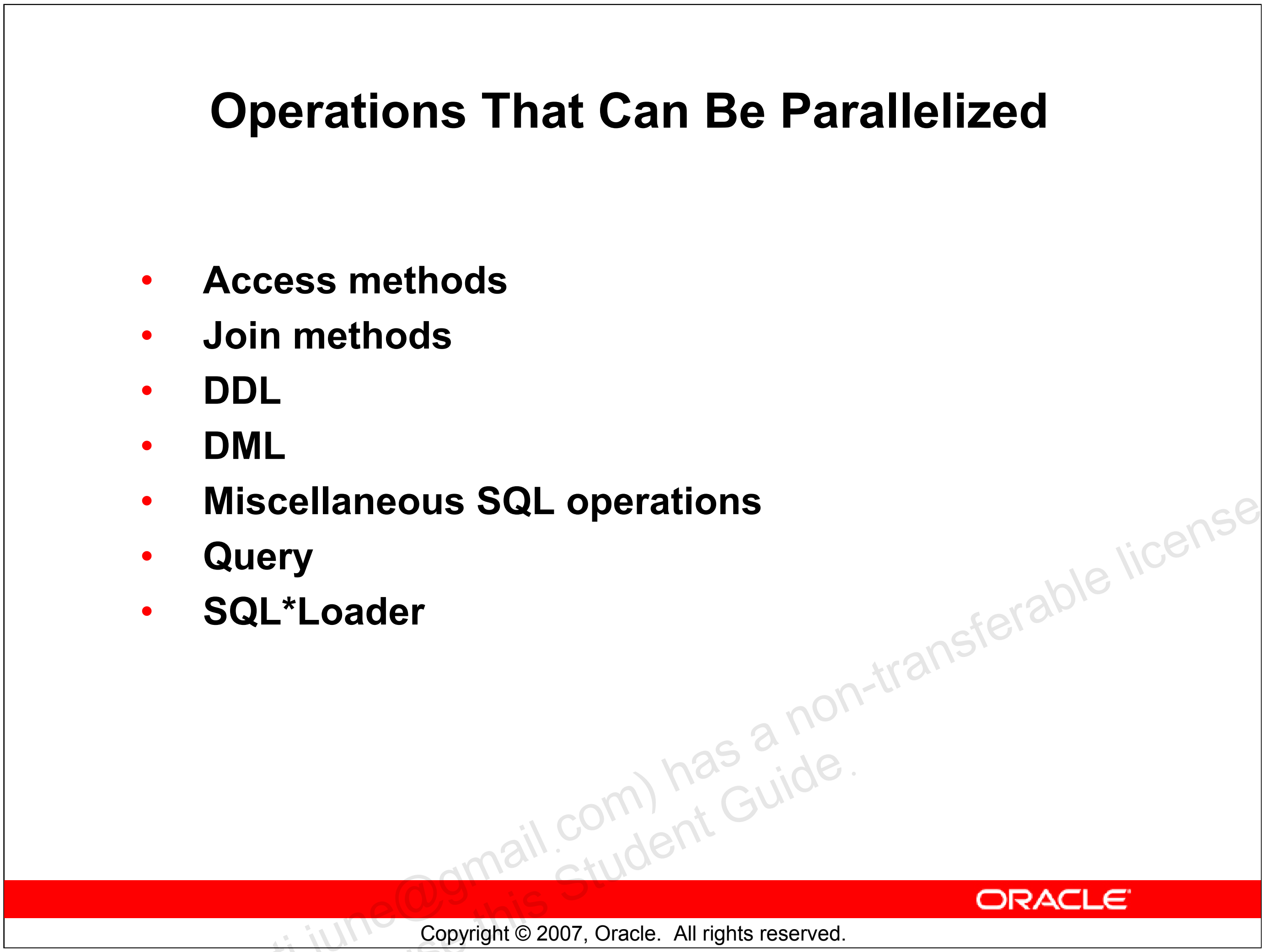
### When to Implement Parallel Execution

The benefits of parallel execution can be seen in DSS and data warehousing environments. OLTP systems can also benefit from parallel execution during batch processing and during schema maintenance operations such as creation of indexes. The average simple data manipulation language (DML) or `SELECT` statements that characterize OLTP applications would not see any benefit from being executed in parallel.

### When Not to Implement Parallel Execution

Parallel execution is not normally useful for:

- Environments in which the typical query or transaction is very short (a few seconds or less). This includes most online transaction systems. Parallel execution is not useful in these environments because there is a cost associated with coordinating the parallel execution servers; for short transactions, the cost of this coordination may outweigh the benefits of parallelism.
- Environments in which the CPU, memory, or I/O resources are already heavily utilized. Parallel execution is designed to exploit additional available hardware resources; if no such resources are available, then parallel execution does not yield any benefits and indeed may be detrimental to performance



## Operations That Can Be Parallelized

- Access methods
- Join methods
- DDL
- DML
- Miscellaneous SQL operations
- Query
- SQL*Loader

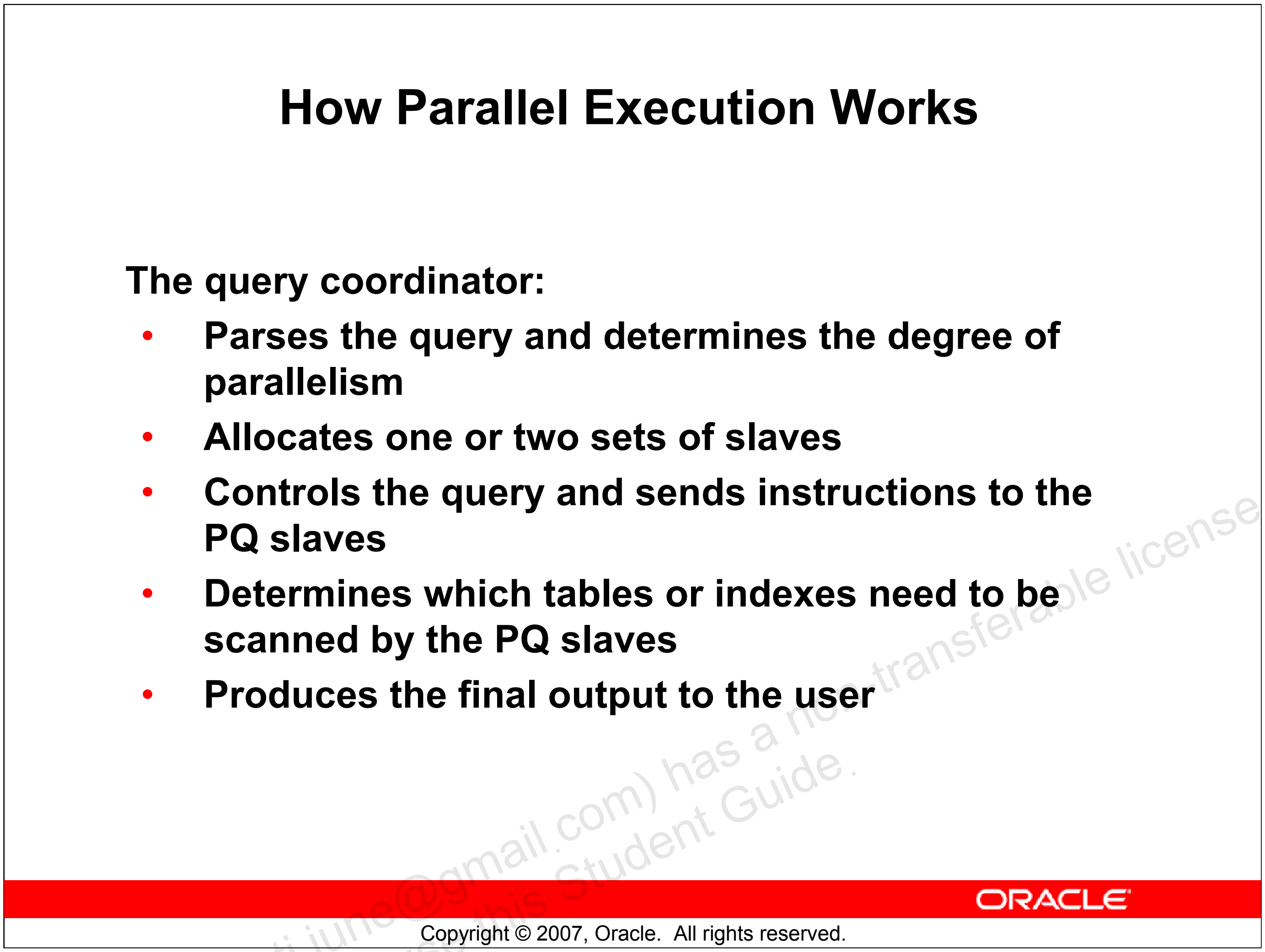
### Operations That Can Be Parallelized

You can use parallel execution for any of the following:

- **Access methods:** Some examples are table scans, index full scans, and partitioned index range scans.
- **Join methods:** Some examples are nested loop, sort merge, hash, and star transformation.
- **DDL statements:** Some examples are `CREATE TABLE AS SELECT`, `CREATE INDEX`, `REBUILD INDEX`, `REBUILD INDEX PARTITION`, and `MOVE/SPLIT/COALESCE PARTITION`. You can normally use parallel data definition language (DDL) where you use regular DDL. There are, however, some additional details to consider when designing your database. One important restriction is that parallel DDL cannot be used on tables with object or LOB columns. All of these DDL operations can be performed in `NOLOGGING` mode for either parallel or serial execution. The `CREATE TABLE` statement for an index-organized table can be parallelized either with or without an `AS SELECT` clause. Different parallelism is used for different operations. The Parallel `CREATE (PARTITIONED) TABLE…` `AS SELECT` and parallel `CREATE (PARTITIONED) INDEX` statements run with a degree of parallelism equal to the number of partitions. Parallel operations require accurate statistics to perform optimally.

### Operations That Can Be Parallelized (continued)

- **DML statements**: Some examples are `INSERT AS SELECT`, updates, deletes, and `MERGE` operations. Parallel DML (parallel insert, update, merge, and delete) uses parallel execution mechanisms to speed up or scale up large DML operations against large database tables and indexes. You can also use `INSERT ... SELECT` statements to insert rows into multiple tables as part of a single DML statement. You can normally use parallel DML where you use regular DML. Although DML normally includes queries, the term *parallel DML* refers only to inserts, updates, merges, and deletes done in parallel.
- **Miscellaneous SQL operations**: Some examples are `GROUP BY`, `NOT IN`, `SELECT DISTINCT`, `UNION`, `UNION ALL`, `CUBE`, and `ROLLUP`, as well as aggregate and table functions.
- **Parallel query**: You can parallelize queries and subqueries in `SELECT` statements, as well as the query portions of DDL statements and DML statements (`INSERT`, `UPDATE`, `DELETE`, and `MERGE`).
- **SQL*Loader:** You can parallelize the use of SQL*Loader where large amounts of data are routinely encountered.



## How Parallel Execution Works

The query coordinator:

- Parses the query and determines the degree of parallelism
- Allocates one or two sets of slaves
- Controls the query and sends instructions to the PQ slaves
- Determines which tables or indexes need to be scanned by the PQ slaves
- Produces the final output to the user

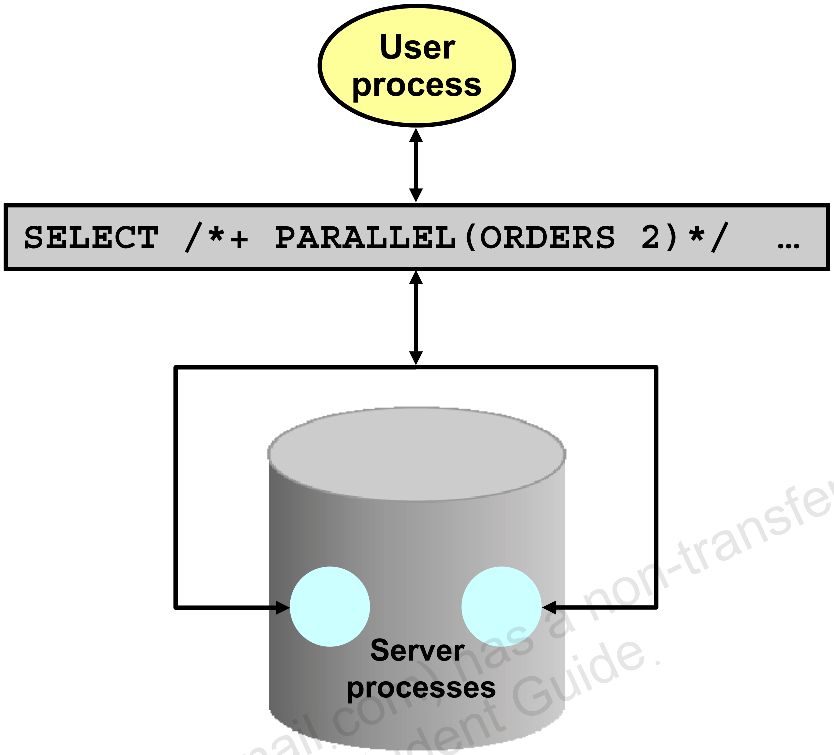
### How Parallel Execution Works

Parallel execution divides the task of executing a SQL statement into multiple small units, each of which is executed by a separate process. The incoming data can be divided into parts (called *granules*). The user shadow process that is going to execute a query in parallel takes on the role as parallel execution coordinator or query coordinator. The query coordinator does the following:

- Parses the query and determines the degree of parallelism
- Allocates one or two sets of slaves (threads or processes)
- Controls the query and sends instructions to the PQ slaves
- Determines which tables or indexes need to be scanned by the PQ slaves
- Produces the final output to the user

At execution time, the coordinator also performs the parts of the plan that execute serially (such as accessing tables in serial if they are small or have no hint or degree of parallelism set). Ranging is also done serially to determine the ranges of keys to be distributed from producer slaves to consumer slaves who are sorting or otherwise must consume specific ranges of rows.

# Degree of Parallelism



### Degree of Parallelism (DOP)

The parallel execution coordinator may enlist two or more of the instance's parallel execution servers to process a SQL statement. The number of parallel execution servers associated with a single operation is known as the *degree of parallelism*. A single operation is a part of a SQL statement such as an order by or full table scan to perform a join on a non-indexed column table. The degree of parallelism applies directly only to intra-operation parallelism. If inter-operation parallelism is possible, the total number of parallel execution servers for a statement can be twice the specified degree of parallelism. No more than two sets of parallel execution servers can run simultaneously. Each set of parallel execution servers may process multiple operations. Only two sets of parallel execution servers need to be active to guarantee optimal inter-operation parallelism.

Parallel execution is designed to effectively use multiple CPUs and disks to answer queries quickly. When multiple users use parallel execution at the same time, it is easy to quickly exhaust available CPU, memory, and disk resources.

The default DOP is used when you ask to parallelize an operation but you do not specify a DOP in a hint or in the definition of a table or index. The default DOP is appropriate for most applications.

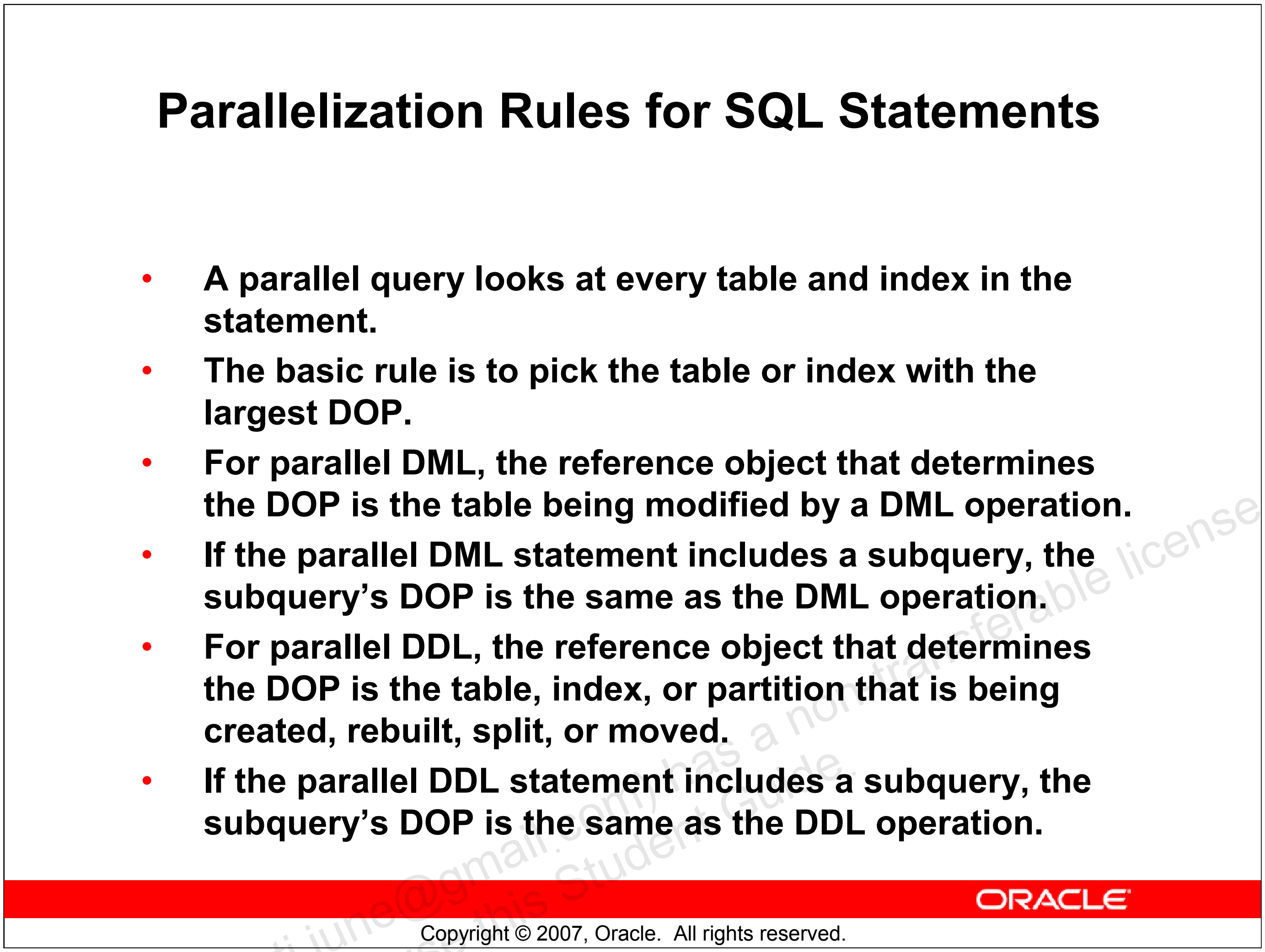
### Degree of Parallelism (continued)

If no parallel hints are used and there is no default degree of parallelism for the table in the dictionary:

- Execution for that table is serial
- When parallelism is enforced with the ALTER SESSION FORCE PARALLEL … command, the DOP for a SQL statement is determined by the value of the parameter CPU_COUNT. The value of CPU_COUNT is, by default, the number of CPUs on the system and the value of the PARALLEL_THREADS_PER_CPU parameter.

However, the actual number of processes that are used is limited by their availability on the requested instances during run time. The PARALLEL_MAX_SERVERS initialization parameter sets an upper limit on the total number of parallel execution servers that an instance can have.

If a minimum fraction of the desired parallel execution servers is not available (specified by the PARALLEL_MIN_PERCENT initialization parameter), a user error is produced. You can retry the query when the system is less busy.



## Parallelization Rules for SQL Statements

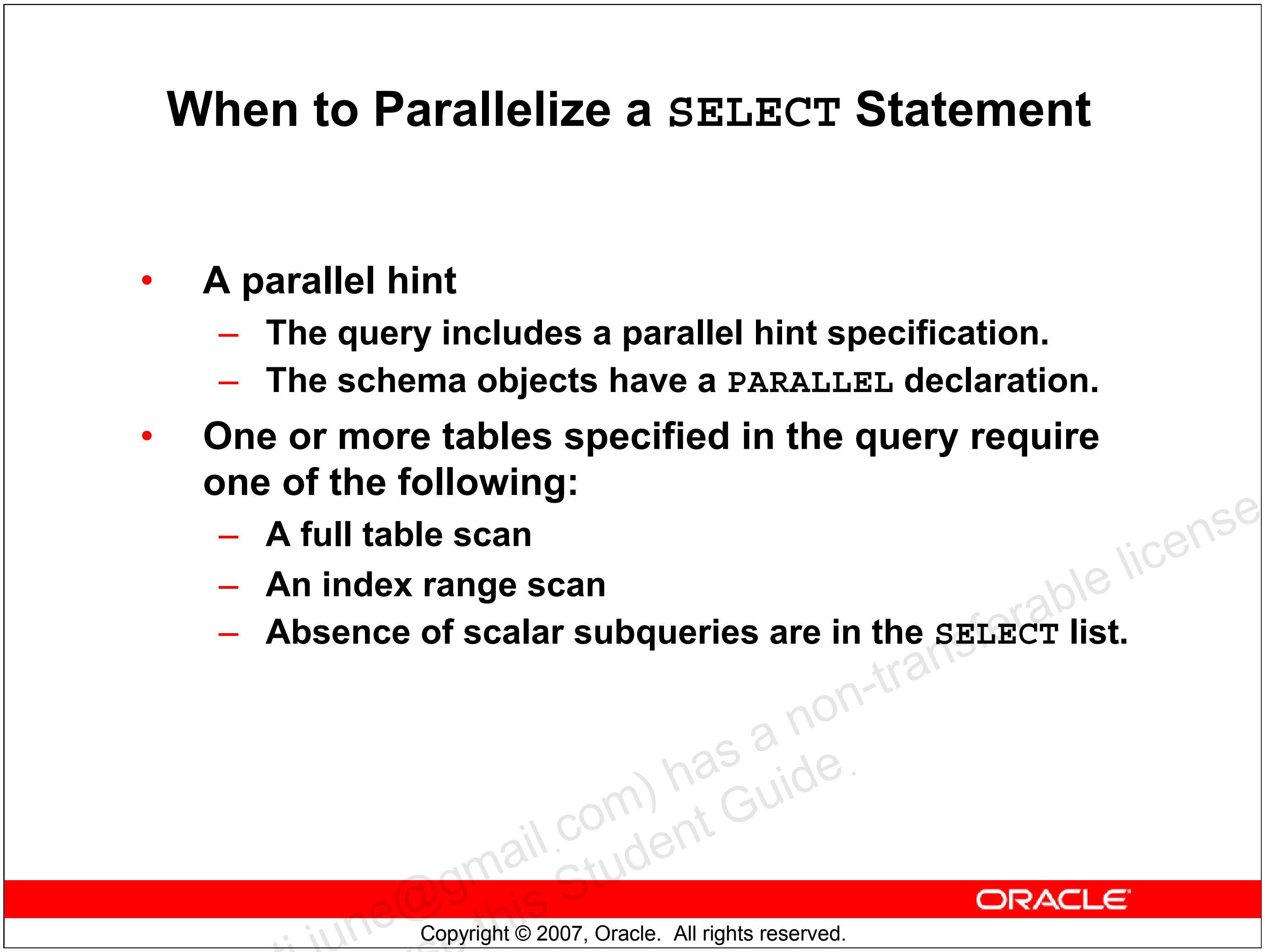
- A parallel query looks at every table and index in the statement.
- The basic rule is to pick the table or index with the largest DOP.
- For parallel DML, the reference object that determines the DOP is the table being modified by a DML operation.
- If the parallel DML statement includes a subquery, the subquery's DOP is the same as the DML operation.
- For parallel DDL, the reference object that determines the DOP is the table, index, or partition that is being created, rebuilt, split, or moved.
- If the parallel DDL statement includes a subquery, the subquery's DOP is the same as the DDL operation.

### Parallelization Rules for SQL Statements

A SQL statement can be parallelized if it includes a parallel hint or if the table or index being operated on has been declared PARALLEL with a CREATE or ALTER statement. In addition, a DDL statement can be parallelized with the PARALLEL clause. However, not all of these methods apply to all types of SQL statements. Parallelization has two components: the decision to parallelize and the DOP. These components are determined differently for queries, DDL operations, and DML operations.

To determine the DOP, the Oracle Server looks at the reference objects:

- Parallel query looks at each table and index in the portion of the query that is being parallelized to determine which is the reference table. The basic rule is to pick the table or index with the largest DOP.
- For parallel DML (INSERT, UPDATE, MERGE, and DELETE), the reference object that determines the DOP is the table being modified by an insert, update, or delete operation. Parallel DML also adds some limits to the DOP to prevent deadlock. If the parallel DML statement includes a subquery, the subquery's DOP is the same as the DML operation.
- For parallel DDL, the reference object that determines the DOP is the table, index, or partition being created, rebuilt, split, or moved. If the parallel DDL statement includes a  subquery, the subquery's DOP is the same as the DDL operation.



## When to Parallelize a SELECT Statement

- A parallel hint
  - The query includes a parallel hint specification.
  - The schema objects have a PARALLEL declaration.
- One or more tables specified in the query require one of the following:
  - A full table scan
  - An index range scan
  - Absence of scalar subqueries are in the SELECT list.

ORACLE

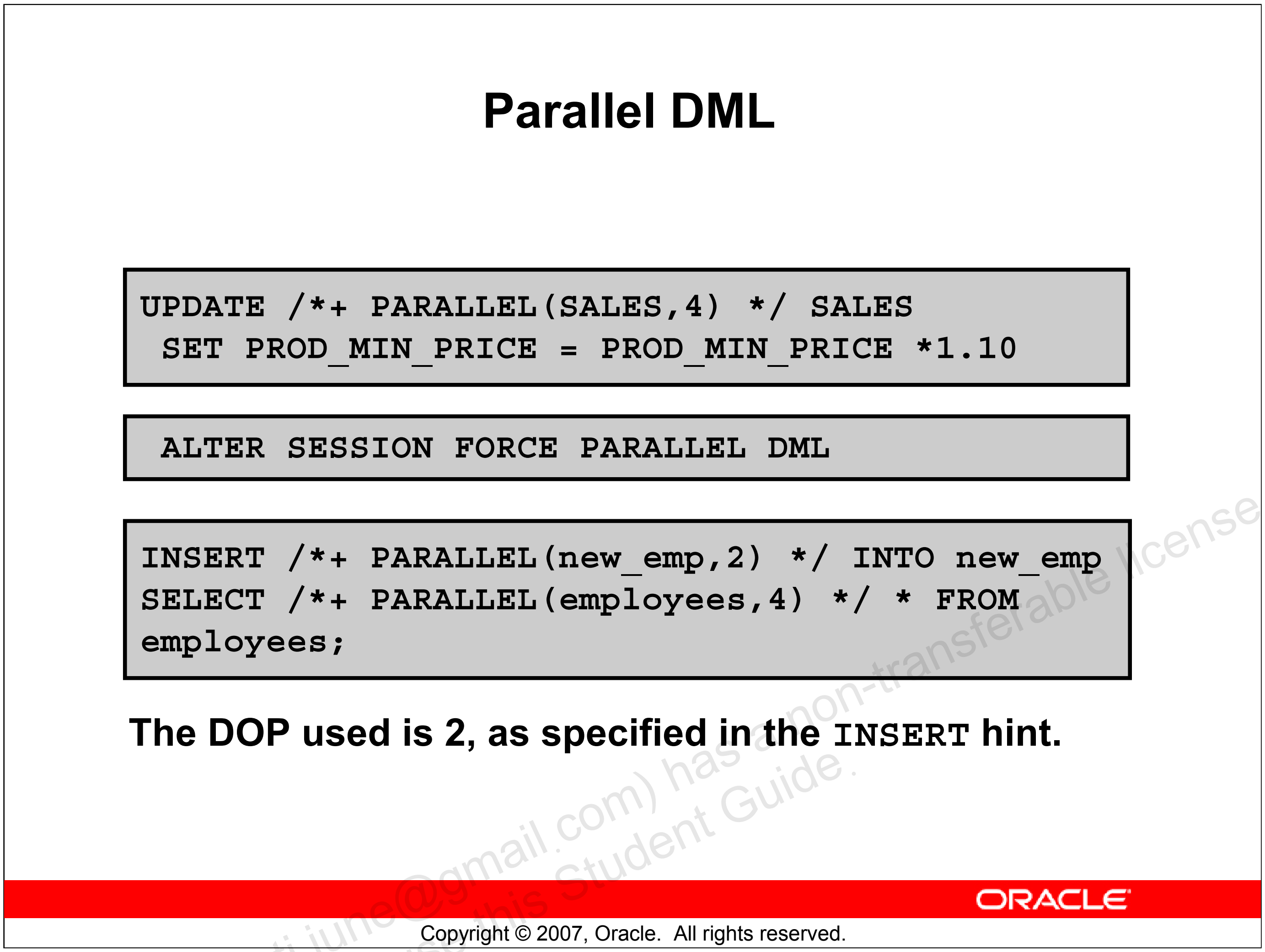
### When to Parallelize a SELECT Statement

A SELECT statement can be parallelized only if the following conditions are satisfied:

- The query includes a parallel hint specification (PARALLEL or PARALLEL_INDEX), or the schema objects that are referred to in the query have a PARALLEL declaration associated with them.
- At least one of the tables specified in the query requires one of the following:
  - A full table scan
  - An index range scan spanning multiple partitions
  - Absence of scalar subqueries are in the SELECT list.

The DOP for a query is determined by the following rules:

- The query uses the maximum DOP taken from all of the table declarations involved in the query and all of the potential indexes that are candidates to satisfy the query (the reference objects). That is, the table or index that has the greatest DOP determines the query's DOP (maximum query directive).
- If a table has both a parallel hint specification in the query and a parallel declaration in its table specification, the hint specification takes precedence over parallel declaration specification.



## Parallel DML

```
UPDATE /*+ PARALLEL(SALES,4) */ SALES
 SET PROD_MIN_PRICE = PROD_MIN_PRICE *1.10
```

```
 ALTER SESSION FORCE PARALLEL DML
```

```
INSERT /*+ PARALLEL(new_emp,2) */ INTO new_emp
SELECT /*+ PARALLEL(employees,4) */ * FROM
employees;
```

**The DOP used is 2, as specified in the INSERT hint.**

### Parallel DML

You have two ways to specify parallel directives for UPDATE, MERGE, and DELETE operations (assuming that PARALLEL DML mode is enabled):

- Use a parallel clause in the definition of the table being updated or deleted (the reference object).
- Use an update, merge, or delete parallel hint in the statement.

Parallel hints are placed immediately after the UPDATE, MERGE, or DELETE keywords in UPDATE, MERGE, and DELETE statements. The hint also applies to the underlying scan of the table being changed.

You can use the ALTER SESSION FORCE PARALLEL DML statement to override parallel clauses for subsequent UPDATE, MERGE, and DELETE statements in a session. Parallel hints in UPDATE, MERGE, and DELETE statements override the ALTER SESSION FORCE PARALLEL DML statement.

### Parallel DML (continued)

The UPDATE or DELETE operation will be parallelized if and only if at least one of the following is true:

- The table being updated or deleted has a PARALLEL specification.
- The PARALLEL hint is specified in the DML statement.
- An ALTER SESSION FORCE PARALLEL DML statement has been issued previously during the session.

If the statement contains subqueries or updatable views, then they may have their own separate parallel hints or clauses. However, these parallel directives do not affect the decision to parallelize UPDATE, MERGE, or DELETE.

The parallel hint or clause on the tables is used by both the query and the UPDATE, MERGE, or DELETE portions to determine parallelism. The decision to parallelize the UPDATE, MERGE, or DELETE portion is made independently of the query portion, and vice versa.

The DOP is determined by the same rules as those used for the queries. Note that in the case of UPDATE and DELETE operations, only the target table to be modified (the only reference object) is involved. Thus, the UPDATE or DELETE parallel hint specification takes precedence over the parallel declaration specification of the target table. In other words, the precedence order is as follows:

MERGE, UPDATE, DELETE hint > session > parallel declaration specification of target table

An INSERT ... SELECT statement parallelizes its INSERT and SELECT operations independently, except for the DOP.

You can specify a parallel hint after the INSERT keyword in an INSERT ... SELECT statement. Because the tables being queried are usually not the same as the table being inserted into, the hint enables you to specify parallel directives specifically for the INSERT operation. You have the following ways to specify parallel directives for an INSERT ... SELECT statement (assuming that PARALLEL DML mode is enabled):

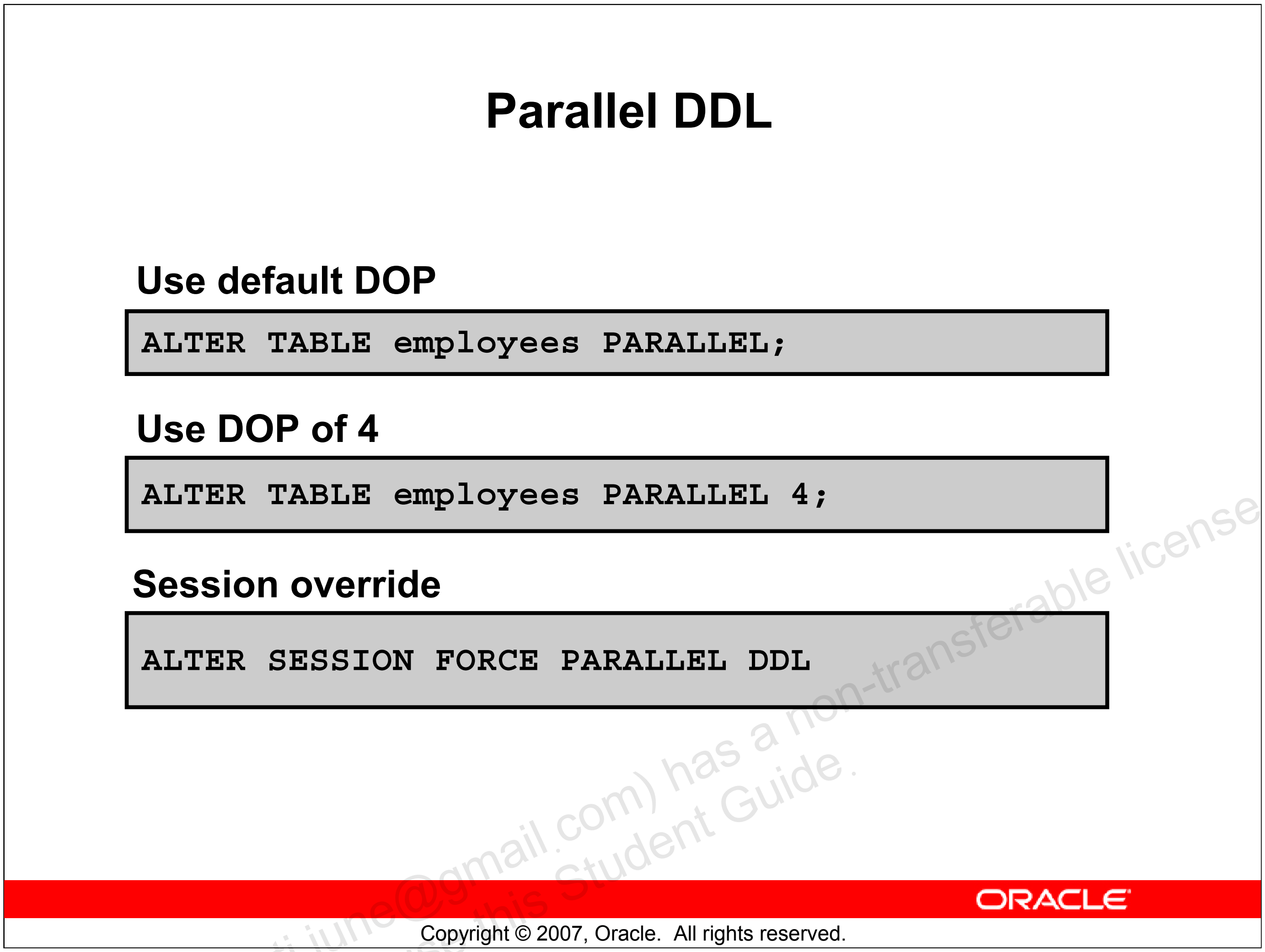
- SELECT parallel hints specified at the statement
- Parallel clauses specified in the definition of tables being selected
- INSERT parallel hint specified at the statement
- Parallel clause specified in the definition of tables being inserted into

You can use the ALTER SESSION FORCE PARALLEL DML statement to override parallel clauses for subsequent INSERT operations in a session. Parallel hints in INSERT operations override the ALTER SESSION FORCE PARALLEL DML statement.

The INSERT operation is parallelized if and only if at least one of the following is true:

- The PARALLEL hint is specified after the INSERT in the DML statement.
- The table being inserted into (the reference object) has a PARALLEL declaration specification.
- An ALTER SESSION FORCE PARALLEL DML statement has been issued previously during the session.

The decision to parallelize the INSERT operation is made independently of the SELECT operation, and vice versa.



## Parallel DDL

Use default DOP

```
ALTER TABLE employees PARALLEL;
```

Use DOP of 4

```
ALTER TABLE employees PARALLEL 4;
```

Session override

```
ALTER SESSION FORCE PARALLEL DDL
```

Copyright © 2007, Oracle. All rights reserved.

### Parallel DDL

DDL operations can be parallelized if a `PARALLEL` clause (declaration) is specified in the syntax. In the case of `CREATE INDEX` and `ALTER INDEX ... REBUILD` or `ALTER INDEX ... REBUILD PARTITION`, the parallel declaration is stored in the data dictionary. You can use the `ALTER SESSION FORCE PARALLEL DDL` statement to override the parallel clauses of subsequent DDL statements in a session.

The DOP is determined by the specification in the `PARALLEL` clause, unless it is overridden by an `ALTER SESSION FORCE PARALLEL DDL` statement. A rebuild of a partitioned index is never parallelized.

Parallel clauses in `CREATE TABLE` and `ALTER TABLE` statements specify table parallelism. If a parallel clause exists in a table definition, it determines the parallelism of DDL statements as well as queries. If the DDL statement contains explicit parallel hints for a table, however, those hints override the effect of parallel clauses for that table. You can use the `ALTER SESSION FORCE PARALLEL DDL` statement to override parallel clauses. This command also has a `DEGREE` option that allows you to override the default degree.

### Parallel CREATE INDEX or ALTER INDEX ... REBUILD

The CREATE INDEX and ALTER INDEX ... REBUILD statements can be parallelized only by a PARALLEL clause or an ALTER SESSION FORCE PARALLEL DDL statement.

ALTER INDEX ... REBUILD can be parallelized only for a nonpartitioned index, but ALTER INDEX ... REBUILD PARTITION can be parallelized by a PARALLEL clause or an ALTER SESSION FORCE PARALLEL DDL statement.

The scan operation for ALTER INDEX ... REBUILD (nonpartitioned), ALTER INDEX ... REBUILD PARTITION, and CREATE INDEX has the same parallelism as the REBUILD or CREATE operation and uses the same DOP. If the DOP is not specified for REBUILD or CREATE, the default value is the number of CPUs.

### Rules for CREATE TABLE AS SELECT

The CREATE TABLE ... AS SELECT statement contains two parts: a CREATE part (DDL) and a SELECT part (query). The Oracle Server can parallelize both parts of the statement. The CREATE part follows the same rules as other DDL operations.

**The query part** of a CREATE TABLE ... AS SELECT statement can be parallelized only if at least one of the following conditions is satisfied:

- The query includes a parallel hint specification (PARALLEL or PARALLEL_INDEX), or the CREATE part of the statement has a PARALLEL clause specification, or the schema objects referred to in the query have a PARALLEL declaration associated with them.
- At least one of the tables specified in the query requires either a full table scan or an index range scan spanning multiple partitions.

The **DOP** for the query part of a CREATE TABLE ... AS SELECT statement is determined by one of the following rules:

- The query part uses the values specified in the PARALLEL clause of the CREATE part.
- If the PARALLEL clause is not specified, the default DOP is the number of CPUs.
- If the CREATE is serial, then the DOP is determined by the query.

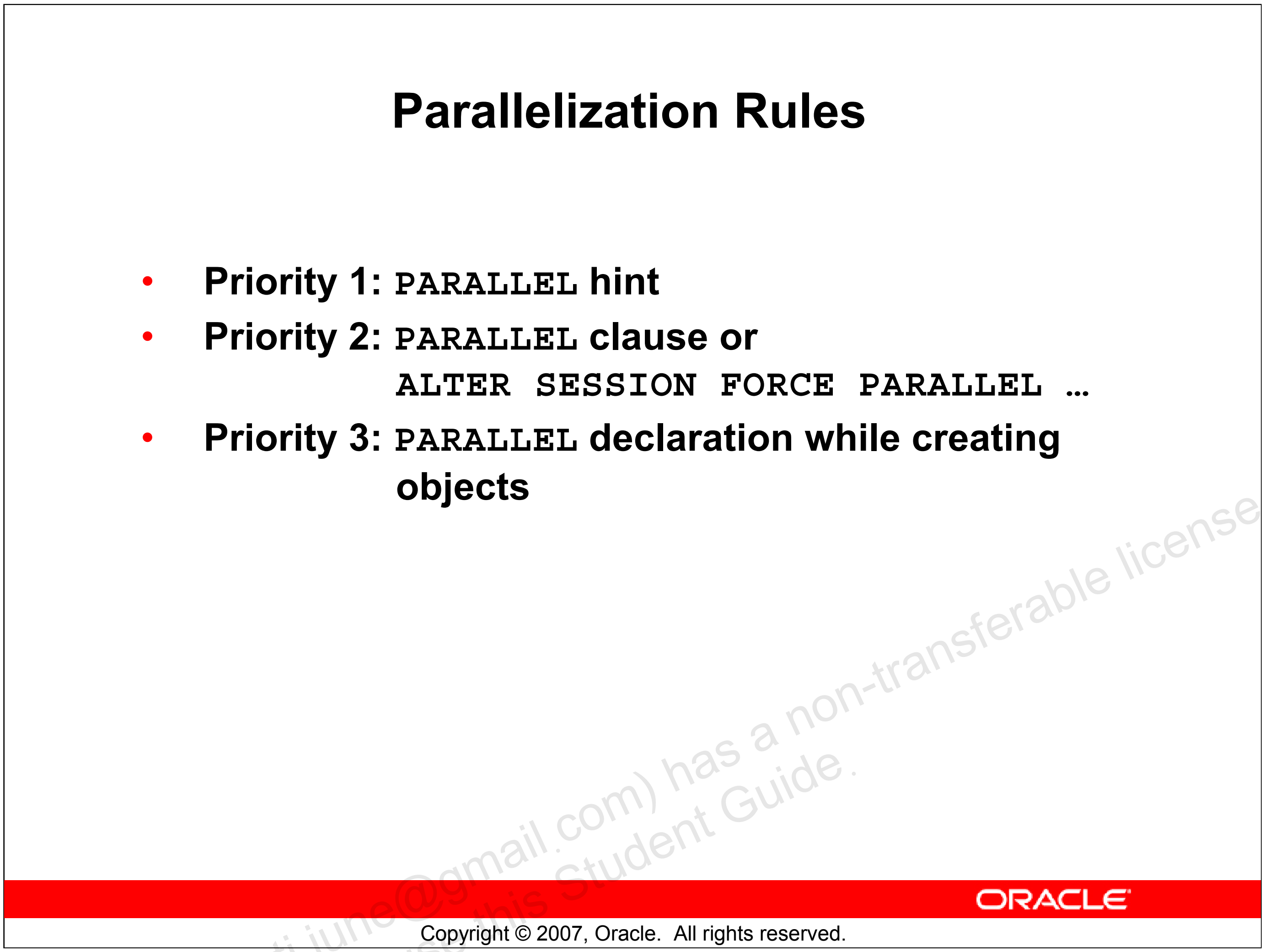
Note that any values specified in a hint for parallelism are ignored.

**The CREATE operation** of CREATE TABLE ... AS SELECT can be parallelized only by a PARALLEL clause or an ALTER SESSION FORCE PARALLEL DDL statement.

When the CREATE operation of CREATE TABLE ... AS SELECT is parallelized, the Oracle Server also parallelizes the scan operation if possible. The scan operation cannot be parallelized if, for example:

- The SELECT clause has a NO_PARALLEL hint
- The operation scans an index of a nonpartitioned table
- When the CREATE operation is not parallelized, the SELECT can be parallelized if it has a PARALLEL hint or if the selected table (or partitioned index) has a parallel declaration

The DOP for the CREATE operation (and for the SELECT operation if it is parallelized) is specified by the PARALLEL clause of the CREATE statement, unless it is overridden by an ALTER SESSION FORCE PARALLEL DDL statement. If the PARALLEL clause does not specify the DOP, the default value is the number of CPUs.



## Parallelization Rules

- Priority 1: `PARALLEL` hint
- Priority 2: `PARALLEL` clause or
  `ALTER SESSION FORCE PARALLEL …`
- Priority 3: `PARALLEL` declaration while creating objects

Copyright © 2007, Oracle. All rights reserved.

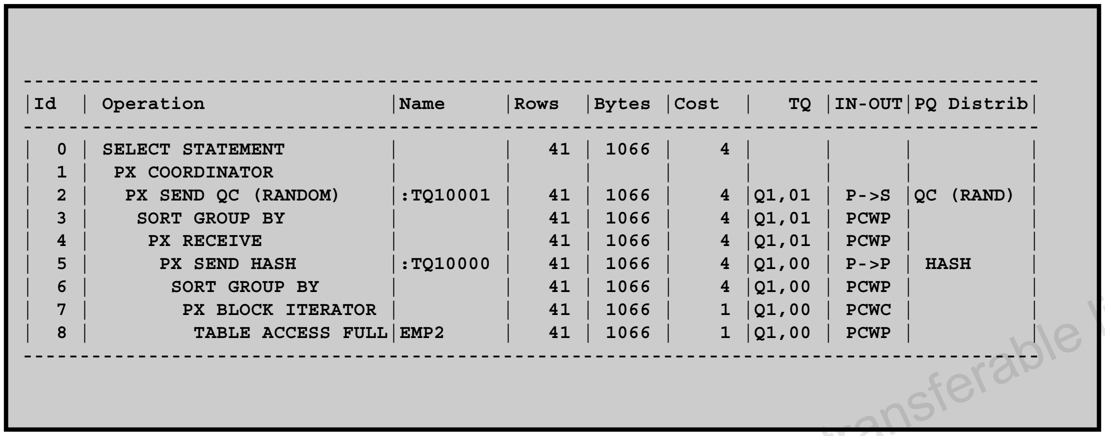
### Parallelization Rules

The slide shows how SQL statements can be parallelized and indicates which methods of specifying parallelism take precedence.

- The priority 1 specification overrides priority 2 and priority 3.
- The priority 2 specification overrides priority 3.

Note: The Oracle Database Resource Manager can limit the degree of parallelism of any operation by a user group, even if hints are used.

# Displaying Parallel Explain Plans



```
----------------------------------------------------------------------------------------------
|Id  | Operation                |Name      |Rows   |Bytes  |Cost   |     TQ|IN-OUT|PQ Distrib|
----------------------------------------------------------------------------------------------
|   0| SELECT STATEMENT         |          |     41|  1066 |      4|       |      |          |
|   1|  PX COORDINATOR          |          |       |       |       |       |      |          |
|   2|   PX SEND QC (RANDOM)    |:TQ10001  |     41|  1066 |      4|Q1,01  | P->S |QC (RAND) |
|   3|    SORT GROUP BY         |          |     41|  1066 |      4|Q1,01  | PCWP |          |
|   4|     PX RECEIVE           |          |     41|  1066 |      4|Q1,01  | PCWP |          |
|   5|      PX SEND HASH        |:TQ10000  |     41|  1066 |      4|Q1,00  | P->P | HASH     |
|   6|       SORT GROUP BY      |          |     41|  1066 |      4|Q1,00  | PCWP |          |
|   7|        PX BLOCK ITERATOR |          |     41|  1066 |      1|Q1,00  | PCWC |          |
|   8|         TABLE ACCESS FULL|EMP2      |     41|  1066 |      1|Q1,00  | PCWP |          |
----------------------------------------------------------------------------------------------
```

## Displaying Parallel Explain Plans

When you use EXPLAIN PLAN with parallel queries, one parallel plan is compiled and executed. This plan is derived from the serial plan by allocating row sources specific to the parallel support in the Query Coordinator (QC) plan. The table queue row sources (PX Send and PX Receive), the granule iterator, and buffer sorts, required by the two slave set PQ model are directly inserted into the parallel plan. This plan is the same plan for all the slaves if executed in parallel or for the QC if executed in serial. You can use the DBMS_XPLAN package to display parallel execution plans. By default, only relevant information is reported by the DISPLAY functions. Parallel information is reported only if the query executes in parallel.

The EXPLAIN plan for the following query is shown in the slide:

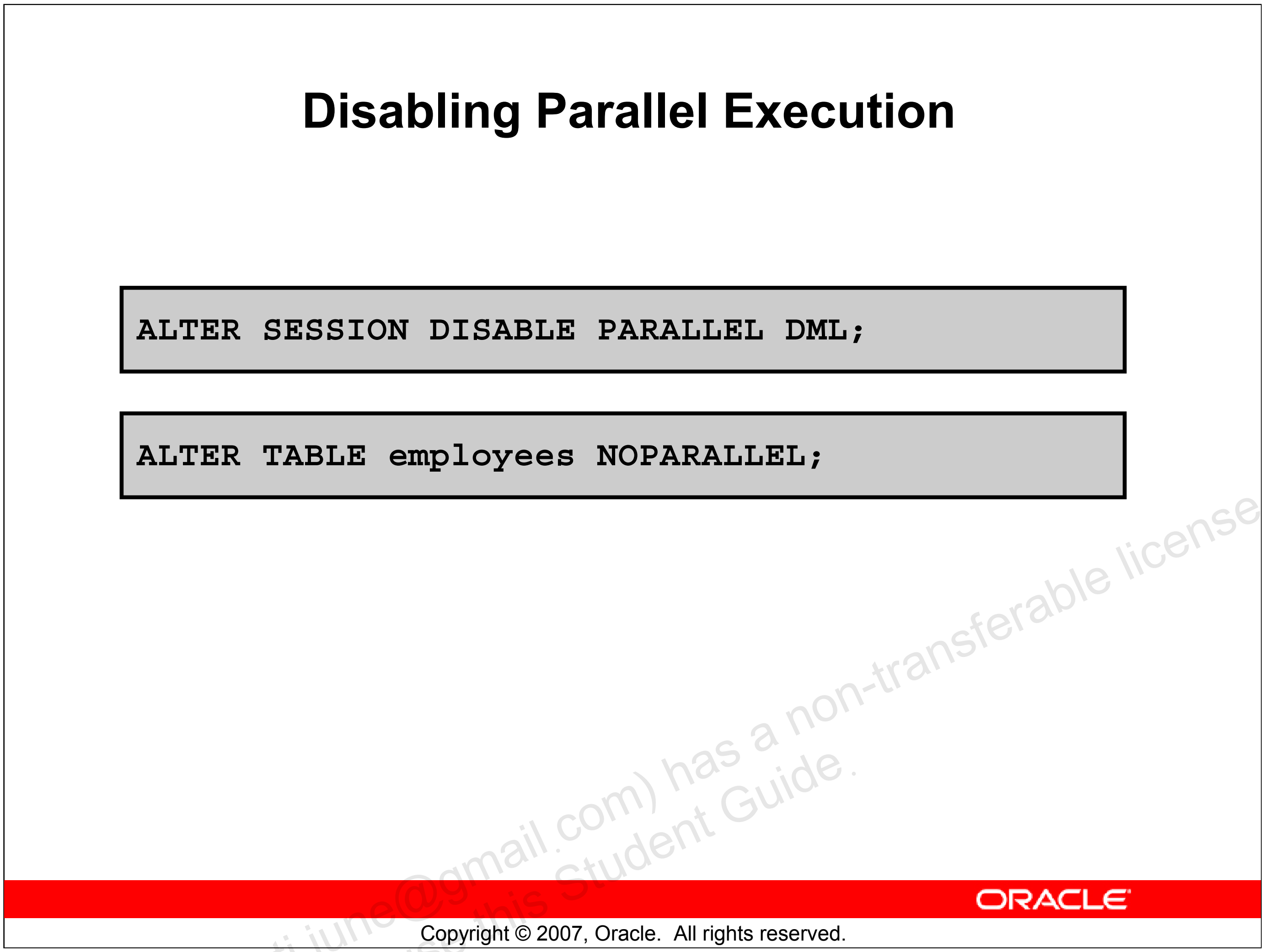
```
CREATE TABLE emp2 AS SELECT * FROM employees;
ALTER TABLE emp2 PARALLEL 2;
EXPLAIN PLAN FOR
  SELECT SUM(salary) FROM emp2 GROUP BY department_id;
SELECT PLAN_TABLE_OUTPUT FROM TABLE(DBMS_XPLAN.DISPLAY());
```

### Displaying Parallel Explain Plans (continued)

When the query is parallel, information related to parallelism is reported: table queue number (TQ column), table queue type (INOUT), and table queue distribution method (PQ Distrib).

The EMP2 table is scanned in parallel by one set of slaves while the aggregation for the GROUP BY is done by the second set. The PX BLOCK ITERATOR row source represents the splitting up of the EMP2 table into pieces so as to divide the scan workload between the parallel scan slaves. The PX SEND and PX RECEIVE row sources represent the pipe that connects the two slave sets as the rows flow up from the parallel scan, are repartitioned through the HASH table queue, and are then read by and aggregated on the top slave set. The PX SEND QC row source represents the aggregated values being sent to the QC in random (RAND) order. The PX COORDINATOR row source represents the QC that controls and schedules the parallel plan appearing below it in the plan tree.

**Note:** The purpose of this example is for you to be able to identify a parallel execution plan. The detailed explanation of each of these terms is beyond the scope of this class.



# Disabling Parallel Execution

```
ALTER SESSION DISABLE PARALLEL DML;
```

```
ALTER TABLE employees NOPARALLEL;
```

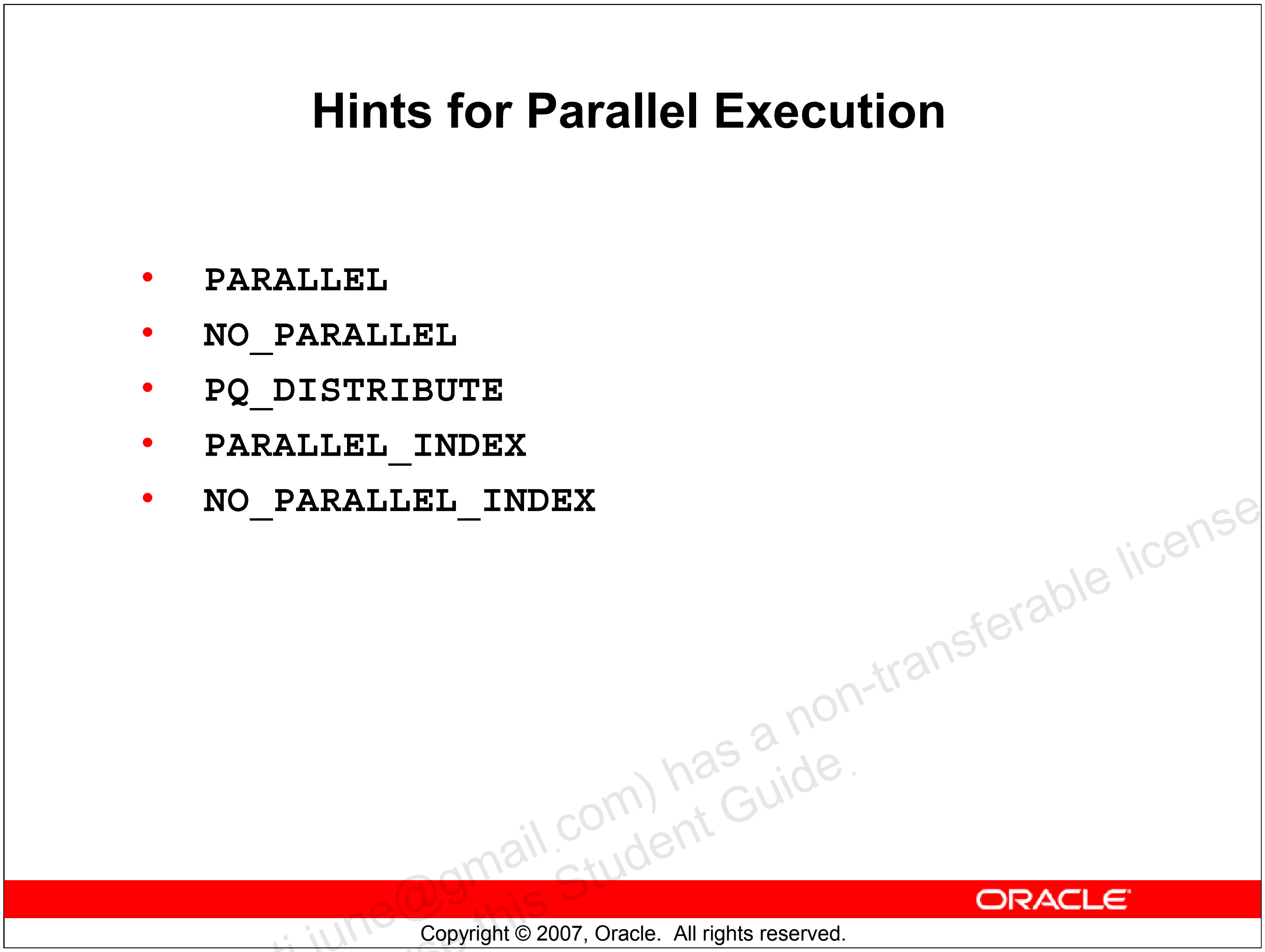
## Disabling Parallel Execution

You disable parallel SQL execution with an ALTER SESSION DISABLE PARALLEL DML|DDL|QUERY statement. All subsequent DML (INSERT, UPDATE, DELETE), DDL (CREATE, ALTER), or query (SELECT) operations are executed serially after such a statement is issued. They are executed serially regardless of any PARALLEL clause that is associated with the statement or any parallel attribute that is associated with the table or indexes involved.

The following statement disables parallel DDL operations:

```
ALTER SESSION DISABLE PARALLEL DDL;
```

You can also use the ALTER statement to change the PARALLEL state of tables and indexes to NOPARALLEL.



## Hints for Parallel Execution

- **PARALLEL**
- **NO_PARALLEL**
- **PQ_DISTRIBUTE**
- **PARALLEL_INDEX**
- **NO_PARALLEL_INDEX**

### Hints for Parallel Execution

**PARALLEL**

The PARALLEL hint lets you specify the desired number of concurrent servers that can be used for a parallel operation. The hint applies to the SELECT, INSERT, UPDATE, and DELETE portions of a statement as well as to the table scan portion.

**NO_PARALLEL**

The NO_PARALLEL hint overrides a PARALLEL specification in the table clause.

**PQ_DISTRIBUTE**

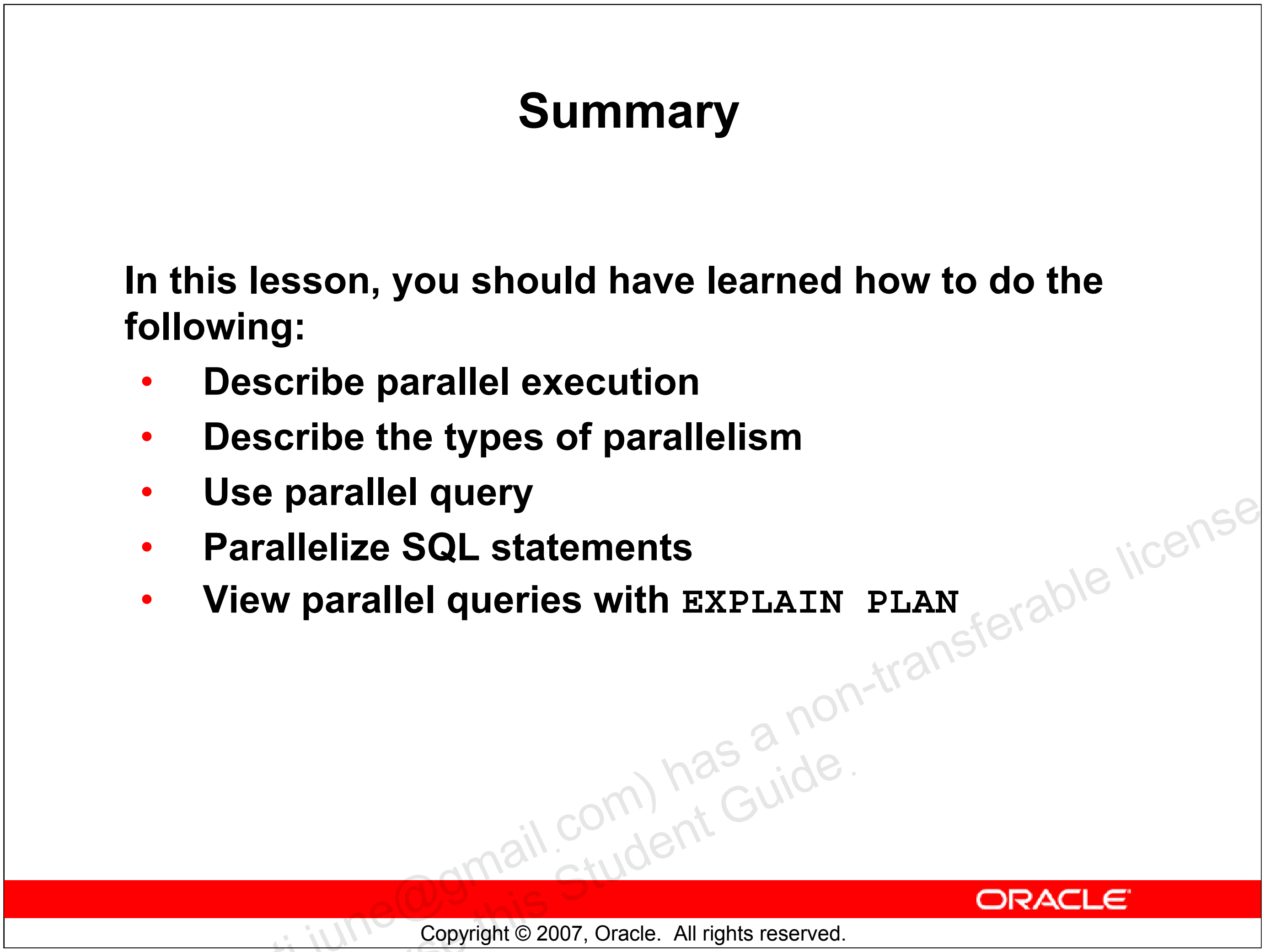
The PQ_DISTRIBUTE hint improves the performance of parallel join operations. The optimizer ignores the distribution hint if both tables are serial.

**PARALLEL_INDEX**

The PARALLEL_INDEX hint specifies the desired number of concurrent servers that can be used to parallelize index range scans for partitioned indexes.

**NO_PARALLEL_INDEX**

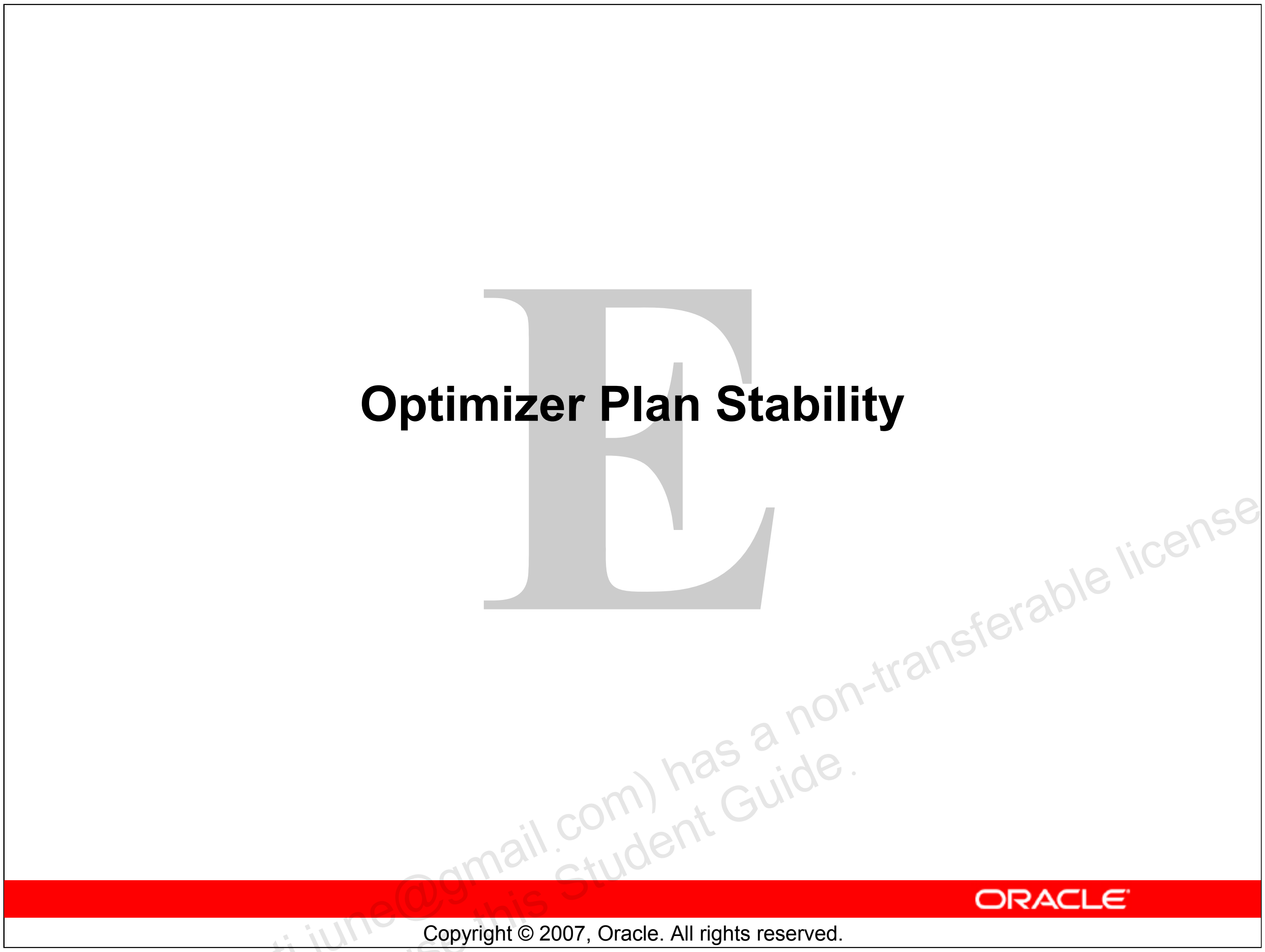
The NO_PARALLEL_INDEX hint overrides a PARALLEL attribute setting on an index to avoid a parallel index scan operation.



## Summary

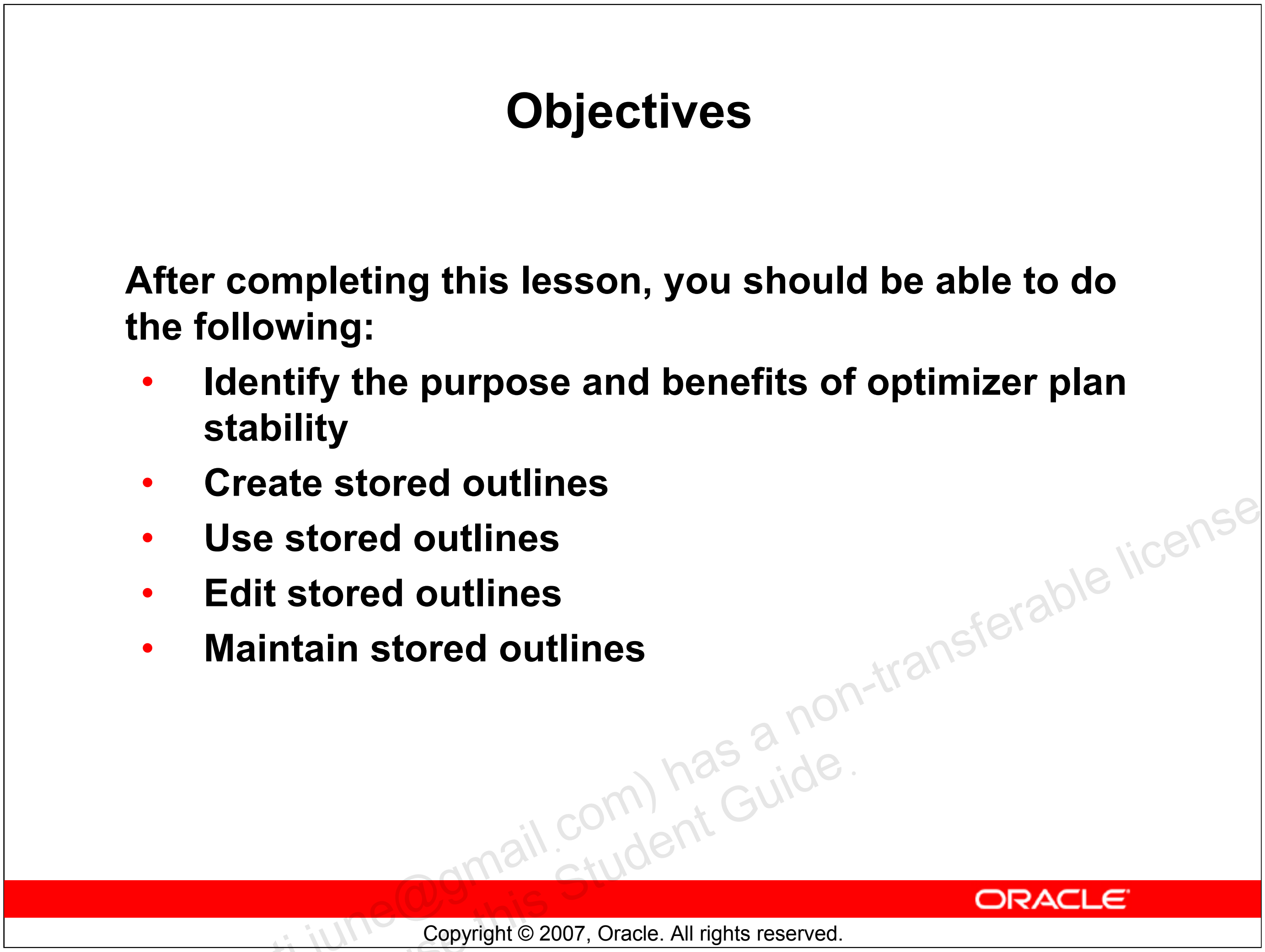
In this lesson, you should have learned how to do the following:

- Describe parallel execution
- Describe the types of parallelism
- Use parallel query
- Parallelize SQL statements
- View parallel queries with `EXPLAIN PLAN`



E

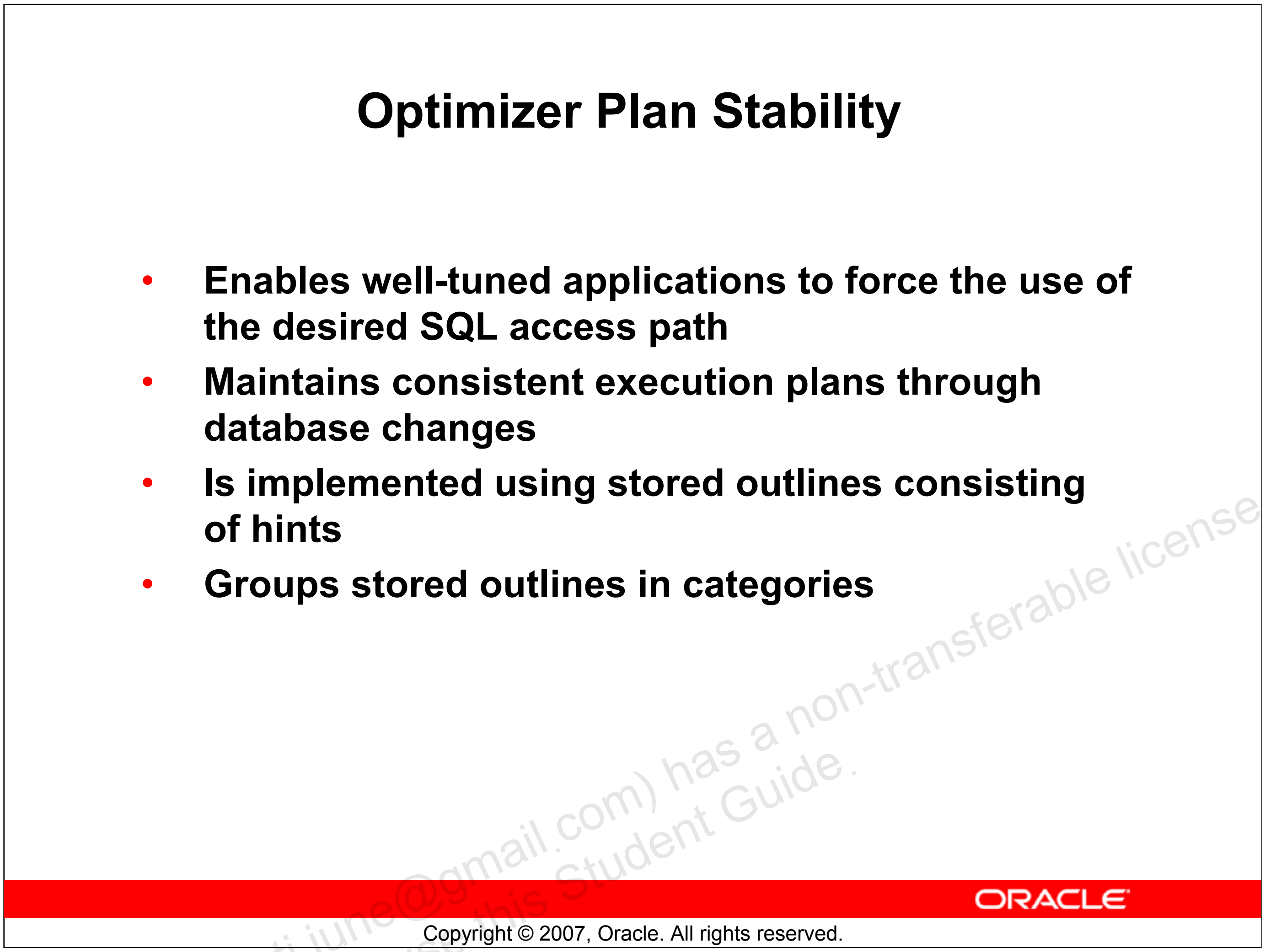
# Optimizer Plan Stability



# Objectives

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

- Identify the purpose and benefits of optimizer plan stability
- Create stored outlines
- Use stored outlines
- Edit stored outlines
- Maintain stored outlines



## Optimizer Plan Stability

- Enables well-tuned applications to force the use of the desired SQL access path
- Maintains consistent execution plans through database changes
- Is implemented using stored outlines consisting of hints
- Groups stored outlines in categories

### Optimizer Plan Stability

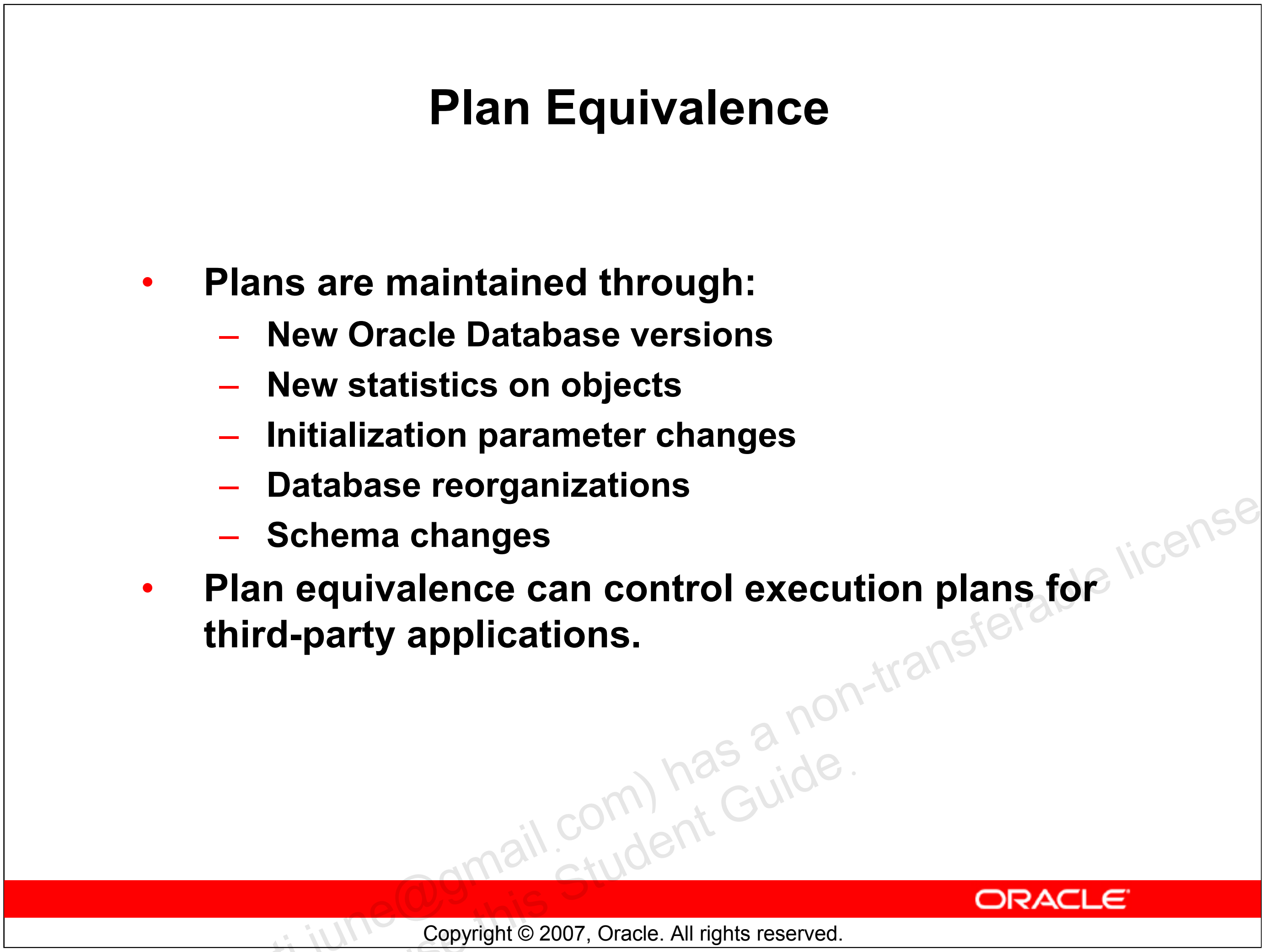
For every query, the optimizer prepares a tree of operations (called an execution plan) that defines the order and methods of operation that the Oracle Server follows to execute the query.

Because the optimizer may have incomplete information, sometimes the best possible plan is not chosen. In these cases, you may find it worthwhile to influence the optimizer’s plan selection by rewriting the SQL statement, by using hints, or by using other tuning techniques.

When satisfied, you may want to ensure that the same tuned plan is generated whenever the same query is recompiled, even when factors that affect optimization have changed.

Oracle Database provides you with a means of stabilizing execution plans for Oracle Database releases or changes, or for other factors that can cause an execution plan to change. You can create a stored outline containing a set of hints that are used by the optimizer to create an execution plan.

Stored outlines can be grouped in named categories. Different categories are useful for creating groups of execution paths for different situations. The same SQL statement may have a stored outline in more than one category. For example, you may want to have an online transaction processing (OLTP) category and a decision support system (DSS) category.



## Plan Equivalence

- Plans are maintained through:
  - New Oracle Database versions
  - New statistics on objects
  - Initialization parameter changes
  - Database reorganizations
  - Schema changes
- Plan equivalence can control execution plans for third-party applications.

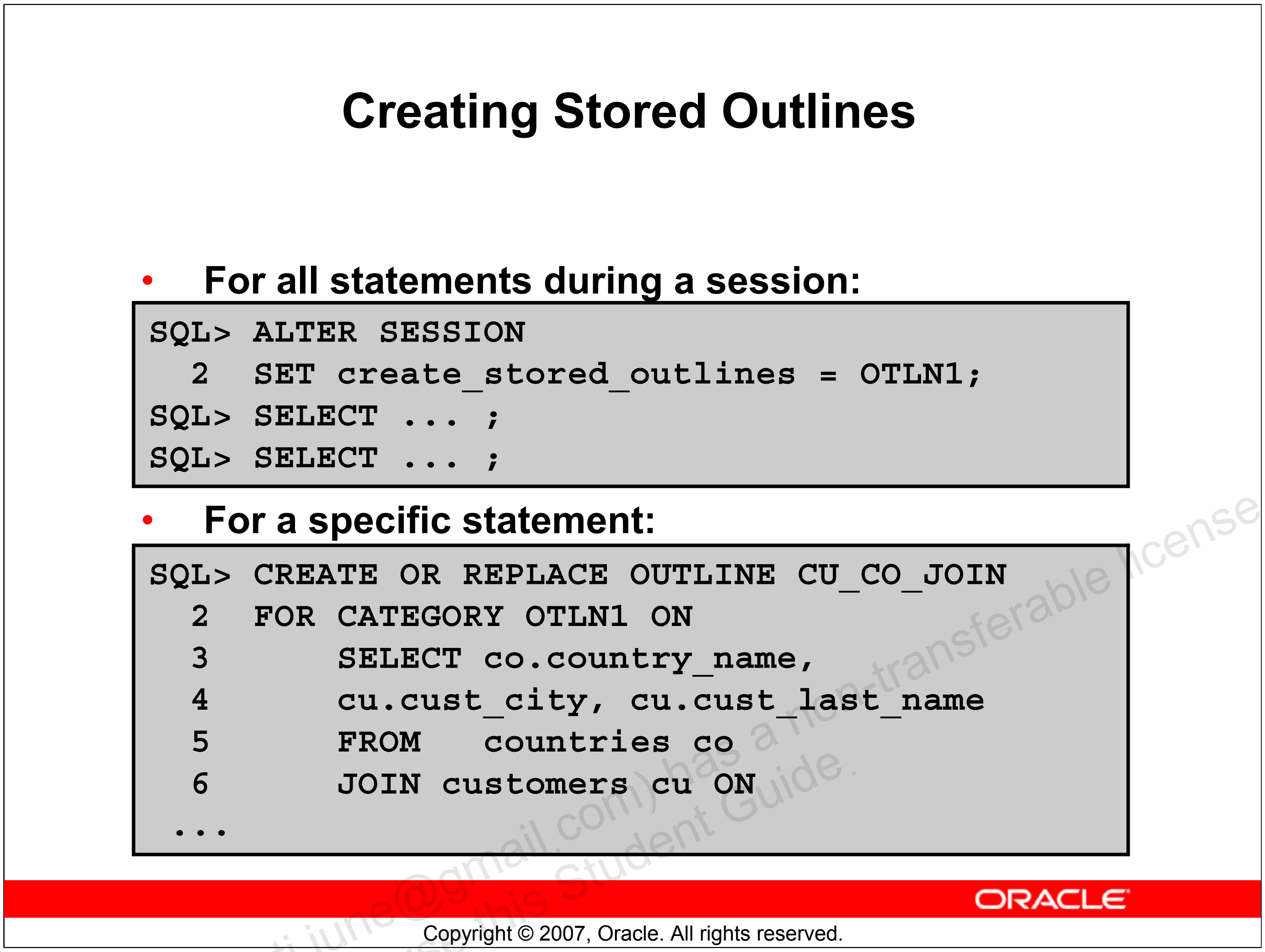
### Plan Equivalence

Plan stability relies on exact textual matching of queries when determining whether a query has a stored outline. However, Oracle Database stored outlines are not case sensitive or white-space sensitive.

Stored outlines rely partially on hints that the optimizer uses to achieve stable execution plans. Therefore, the degree to which a plan remains equivalent depends on the capabilities of the hints that the plan uses. The execution steps included in a stored outline include row access methods, join order, join methods, distributed accesses, and view/subquery merging. Distributed access does not include the execution plan on the remote node.

### Plan Stability

These plans are maintained through many types of database and instance changes. Therefore, if you develop applications for mass distribution, you can use stored outlines to ensure that all your customers access the same execution plans. For example, if adding an index changes the schema, the execution plans do not change. You must generate new stored outlines, or disable the plan stability feature, to benefit from the new index.



## Creating Stored Outlines

- For all statements during a session:

```
SQL> ALTER SESSION
  2  SET create_stored_outlines = OTLN1;
SQL> SELECT ... ;
SQL> SELECT ... ;
```

- For a specific statement:

```
SQL> CREATE OR REPLACE OUTLINE CU_CO_JOIN
  2  FOR CATEGORY OTLN1 ON
  3      SELECT co.country_name,
  4      cu.cust_city, cu.cust_last_name
  5      FROM   countries co
  6      JOIN customers cu ON
 ...
```

### Creating Stored Outlines

You can create outlines for a session, or you can create them for specific SQL statements. Outlines can derive their input from both the rule-based optimizer and the cost-based optimizer. However, the Oracle optimizer can use outlines only when using the cost-based optimizer because outlines rely on hints.

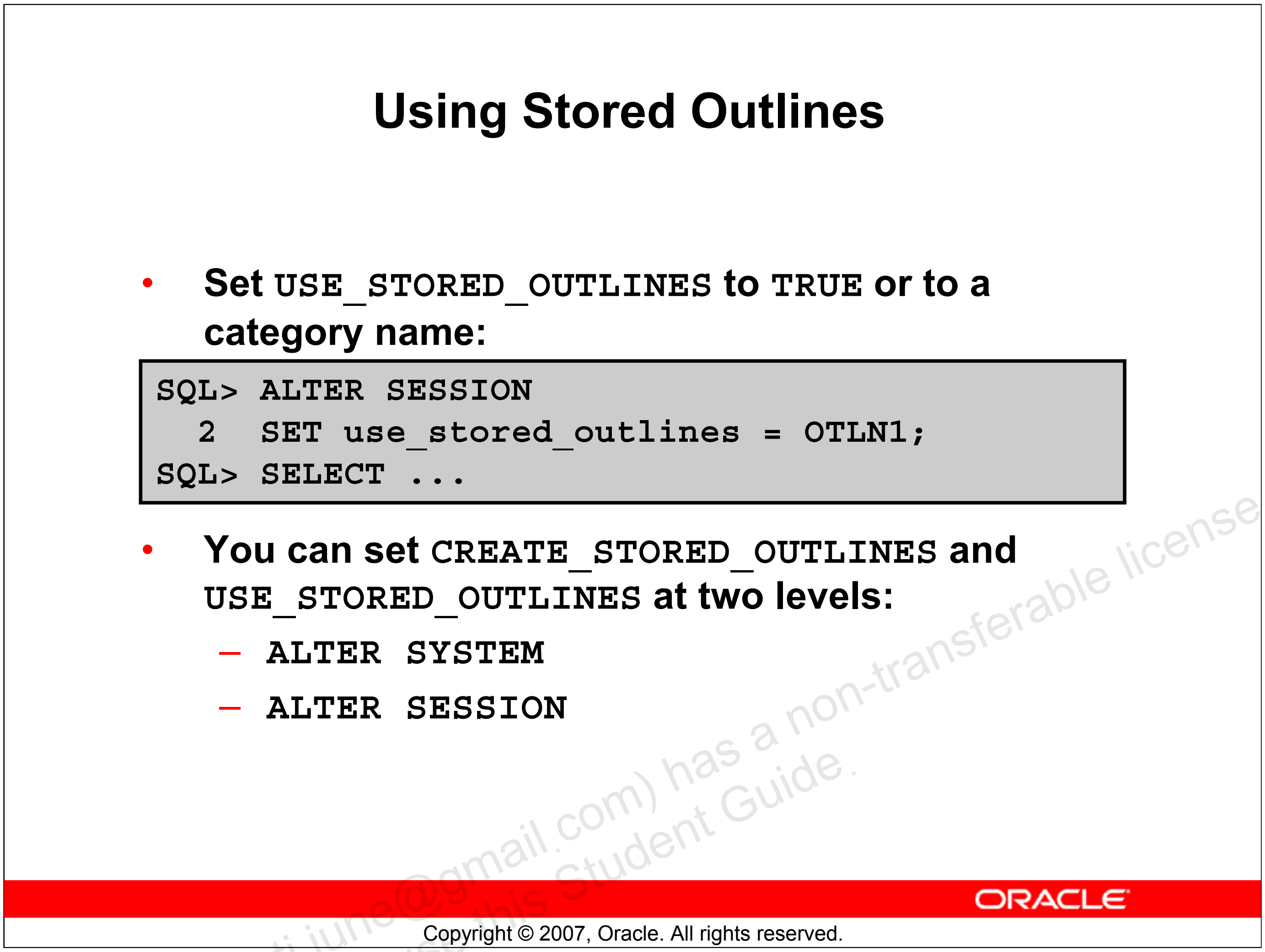
**`CREATE_STORED_OUTLINES` Parameter**

The Oracle Server creates stored outlines automatically when you set this parameter to `TRUE` or to a category name. When the parameter is set to `TRUE`, the `DEFAULT` category is used. When the parameter is activated, the Oracle Server creates outlines for all executed SQL statements. You can deactivate this by setting the parameter to `FALSE`. When you use this parameter, the Oracle Server generates the names for each stored outline.

**`CREATE OUTLINE` Command**

You can also create stored outlines for specific statements by using the `CREATE OUTLINE` command. One advantage of using this command is that you can name stored outlines. In the example, the `CREATE OUTLINE` command assigns the category `OTLN1` to the stored outline.

If you omit a category name when generating outlines, they are placed in the `DEFAULT` category.



# Using Stored Outlines

- Set `USE_STORED_OUTLINES` to `TRUE` or to a category name:

```
SQL> ALTER SESSION
  2  SET use_stored_outlines = OTLN1;
SQL> SELECT ...
```

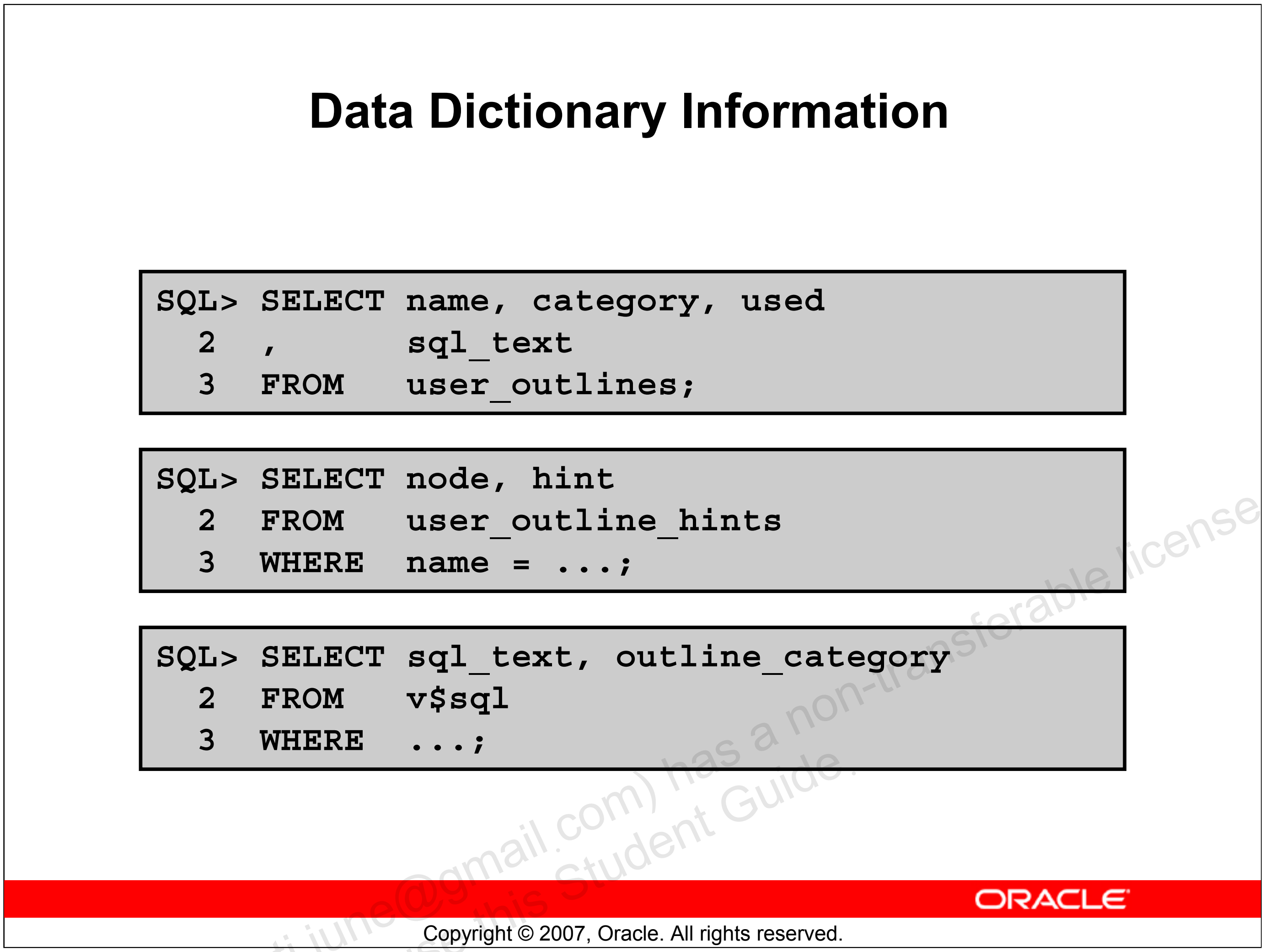
- You can set `CREATE_STORED_OUTLINES` and `USE_STORED_OUTLINES` at two levels:
  - `ALTER SYSTEM`
  - `ALTER SESSION`

## Using Stored Outlines

To use a stored outline for a particular SQL statement:

- USE_STORED_OUTLINES must be set to TRUE or a category name:
  - If it is set to TRUE, then outlines from the DEFAULT category are used.
  - If it is set to a category name, then the outlines from that category are used. If there is no matching outline in that category but there is one in the DEFAULT category, then that outline is used.
- The statement must match the text of the statement in the outline. They are compared using the same method for comparing cursors in the shared pool. This means that hints in the outline must be used in the statement text to cause a match. Values in bind variables do not need to match.

**Note:** Setting CURSOR_SHARING to FORCE or SIMILAR prevents any outlines that are generated with literals from being used if they were generated with CURSOR_SHARING set to EXACT. To use stored outlines with CURSOR_SHARING=FORCE or SIMILAR, the outlines must be generated with CURSOR_SHARING set to FORCE or SIMILAR and with the CREATE_STORED_OUTLINES parameter.



## Data Dictionary Information

```
SQL> SELECT name, category, used
  2  ,      sql_text
  3  FROM   user_outlines;
```

```
SQL> SELECT node, hint
  2  FROM   user_outline_hints
  3  WHERE  name = ...;
```

```
SQL> SELECT sql_text, outline_category
  2  FROM   v$sql
  3  WHERE  ...;
```

### Data Dictionary Information

Information about stored outlines can be obtained from the USER_OUTLINES and USER_OUTLINE_HINTS data dictionary views. The column descriptions of these two views are provided in the following two tables:

**USER_OUTLINES**

| | |
|---|---|
| NAME | Name of the outline |
| CATEGORY | User-defined name of the category to which this outline belongs |
| USED | Outline usage indication (USED, UNUSED, or UNDEFINED) |
| TIMESTAMP | Time stamp of outline creation |
| VERSION | Oracle Database version that created the outline |
| SQL_TEXT | SQL text of the query, including hints that were part of the original statement |

### Data Dictionary Information (continued)

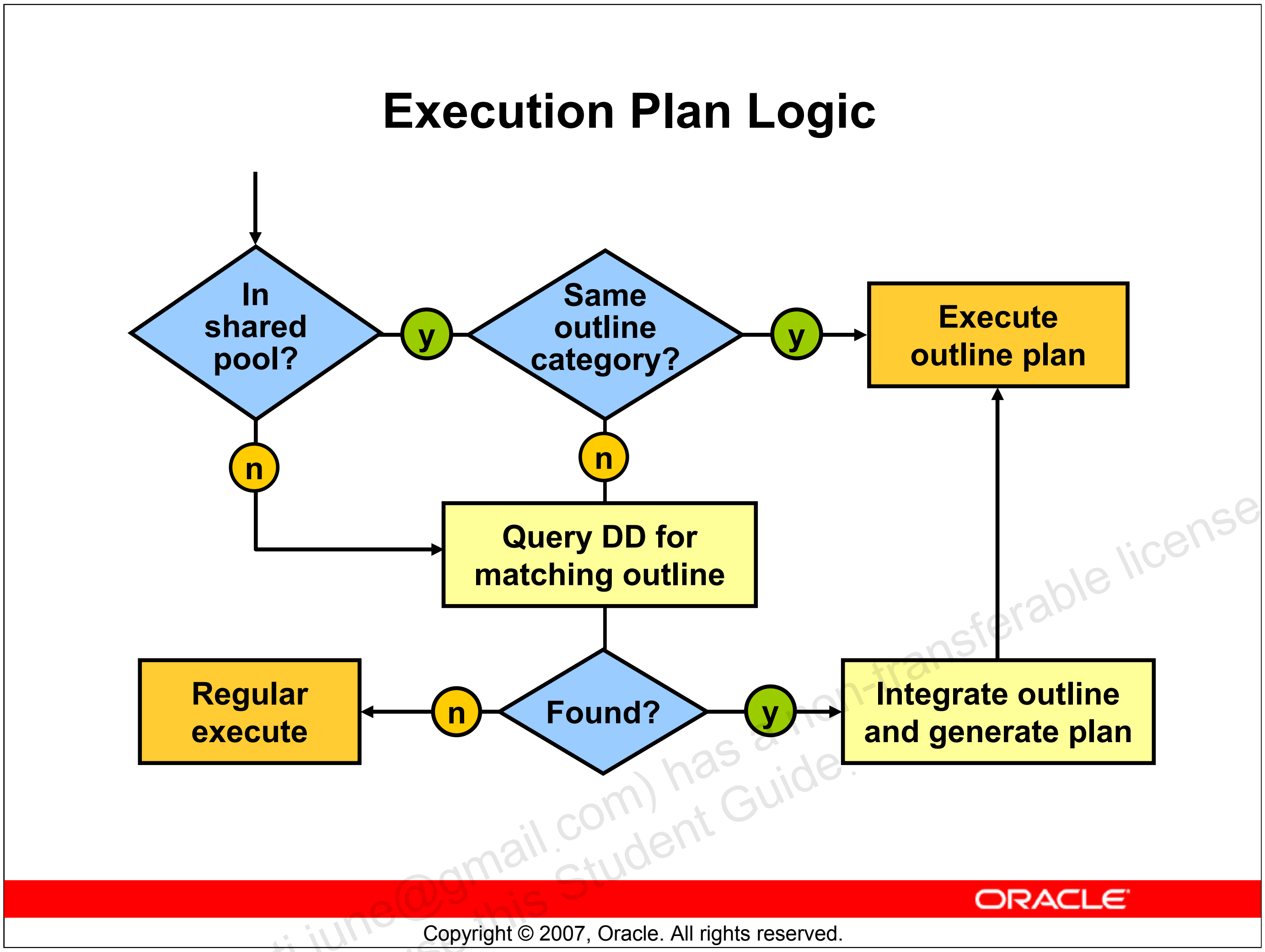
**USER_OUTLINE_HINTS**

| | |
|---|---|
| NAME | Name of the outline |
| NODE | ID of the query or subquery to which the hint applies (The top-level query is labeled 1; subqueries start with 2.) |
| JOIN_POS | Position of the table in the join order (The value is 0 for all hints except access method hints, which identify a table to which the hint and the join position apply.) |
| HINT | Text of the hint |

You can verify that an outline is being used with the V$SQL view. Query the OUTLINE_CATEGORY column in conjunction with the SQL statement. If an outline was applied, this column contains the category to which the outline belongs. Otherwise, it is NULL. The OUTLINE_SID column tells you if this particular cursor is using a public outline (value is 0) or a private outline (the session SID of the corresponding session using it).

Here is an example:

```
SELECT OUTLINE_CATEGORY, OUTLINE_SID
 FROM V$SQL
 WHERE SQL_TEXT LIKE 'SELECT COUNT(*) FROM emp%';
```



## Execution Plan Logic

### Execution Plan Logic

To determine a SQL statement's execution plan, the Oracle optimizer uses the following logic:

- The statement is compared to statements in the shared pool for matching text and outline category.
- If no matching statement is found, the data dictionary is queried for a matching outline.
- If a matching outline is found, the Oracle optimizer integrates the outline into the statement and creates the execution plan.
- If no outline is found, the statement is executed using normal (nonoutline) methods.

If an outline specifies the use of an object that cannot be used, the statement does not use the hint. For example, it references an index that no longer exists. To verify that a stored outline is being used, the explain plan for a statement must be compared when running with and without the `USE_STORED_OUTLINES` set.

**Note:** Cursor sharing can affect the usage of stored outlines.

# Maintaining Stored Outlines

- Use DBMS_OUTLN to:
  - Drop unused outlines
  - Drop categories of outlines
  - Rename a category
- Use ALTER OUTLINE to:
  - Rename an outline
  - Rebuild an outline
  - Change the category of an outline
- Outlines are stored in the OUTLN schema.

## Maintaining Stored Outlines

Use procedures in the DBMS_OUTLN package to manage stored outlines and their categories:

| Procedure | Description |
|---|---|
| DROP_UNUSED | Drops outlines that have not been used since they were created |
| DROP_BY_CAT | Drops outlines assigned to the specified category name |
| UPDATE_BY_CAT | Reassigns outlines from one category to another |

You can alter individual outlines by using the ALTER OUTLINE command and drop them by using the DROP OUTLINE command.

You can export and import plans by exporting the schema OUTLN, where all outlines are stored. Outlines can be queried from tables in the OUTLN schema:

- OL$ contains the outline name, category, create time stamp, and the text of the statement.
- OL$HINTS contains the hints for the outlines in OL$.

Also, there are equivalent data dictionary views: DBA_OUTLINES and DBA_OUTLINE_HINTS.

**Note:** Because the user OUTLN is automatically created with password OUTLN at the time of database creation, this password should be changed for security reasons.

# Maintaining Stored Outlines

```
SQL> BEGIN
  2     dbms_outln.drop_unused;
  3     dbms_outln.update_by_cat
  4       ('default','otln1');
  5     dbms_outln.drop_by_cat('otln1');
  6   END;
```

## Maintaining Stored Outlines (continued)

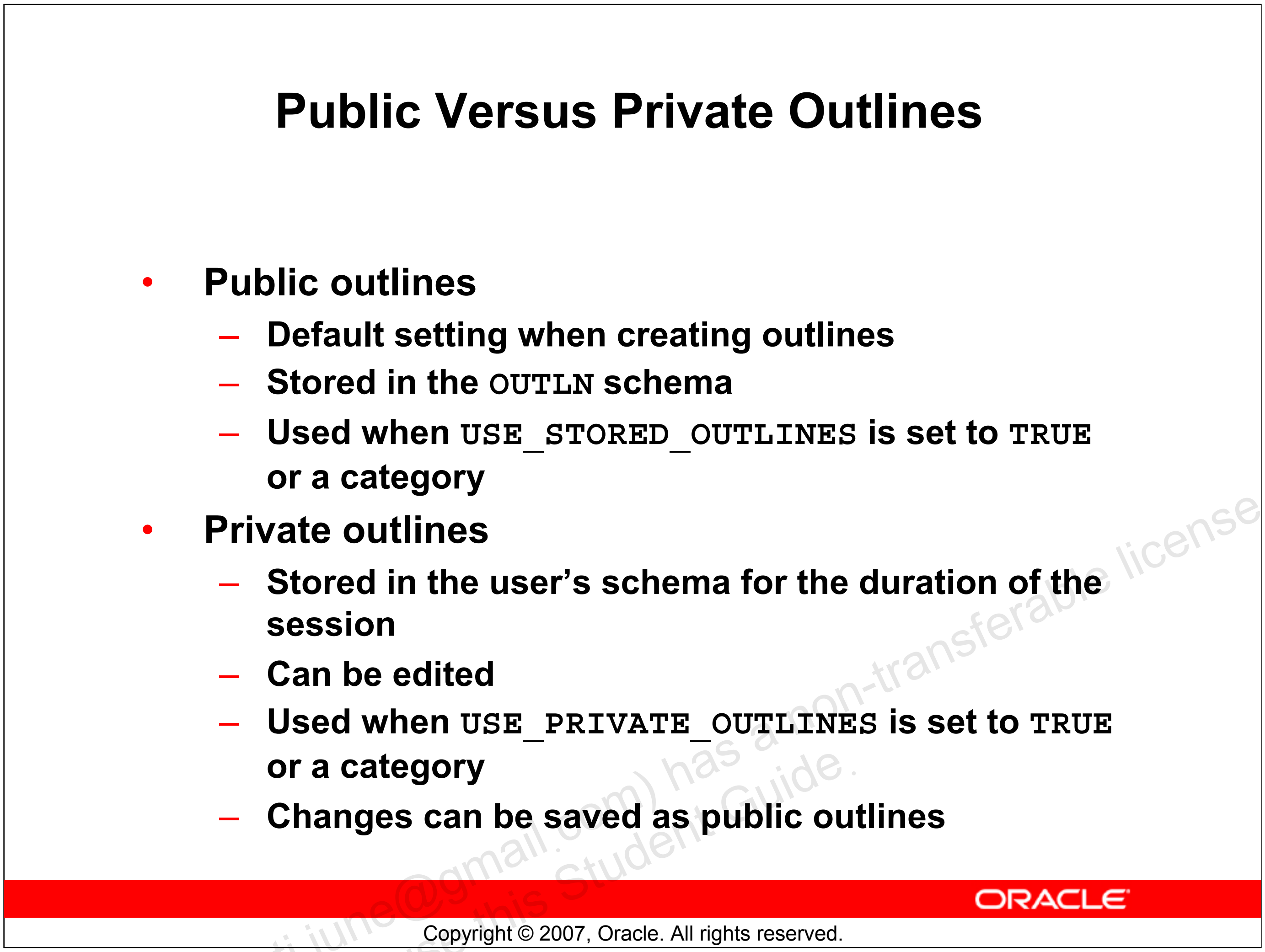
The DROP_UNUSED procedure does not accept arguments and automatically drops all outlines that have never been used since they were created.

The UPDATE_BY_CAT procedure example moves all outlines from the DEFAULT category to the OTLN1 category. For example, this procedure is useful to move a set of outlines from an experimental (test environment) category into a production category.

The DROP_BY_CAT procedure example purges all outlines that belong to the OTLN1 category in a single call.

**Export and Import Stored Outlines**

By exporting and importing the OUTLN schema, you can distribute outlines to other databases (for example, to all departments that use a certain application). This approach ensures execution plan consistency at all sites.



## Public Versus Private Outlines

- Public outlines
  - Default setting when creating outlines
  - Stored in the `OUTLN` schema
  - Used when `USE_STORED_OUTLINES` is set to `TRUE` or a category
- Private outlines
  - Stored in the user's schema for the duration of the session
  - Can be edited
  - Used when `USE_PRIVATE_OUTLINES` is set to `TRUE` or a category
  - Changes can be saved as public outlines

### Public Versus Private Outlines

**Public outlines:** In the Oracle Database, all outlines are public objects that are indirectly available to all system users for whom the USE_STORED_OUTLINES configuration parameter setting applies. Outline data resides in the OUTLN schema that can be thought of as an extension to the SYS schema, in the sense that it is maintained by the system only. To avoid security and integrity issues that are associated with outline data, you are discouraged from manipulating this data directly. Outlines continue to be public by default, and only public outlines are generally available to the user community.

**Private outlines:** The Oracle Database supports private outlines to aid in outline editing. A private outline is an outline that is seen only in the current session and whose data resides in the current parsing schema. Because the outline data for a private outline is stored directly in the user's schema, users are given the opportunity to manipulate the outline data directly through DML in whatever way they choose. Any changes made to such an outline are not seen by any other session on the system, and applying a private outline to the compilation of a statement can be done only in the current session through a new session parameter. Only when a user explicitly chooses to save edits back to the public area do the rest of the users see them. An outline clone is a private outline that has been created by copying data from an existing outline.

## Outline Editing: Overview

- Stored outlines can be edited.
- Users can tune execution plans without having to change the application.
- This is possible by editing the content of the saved plan.

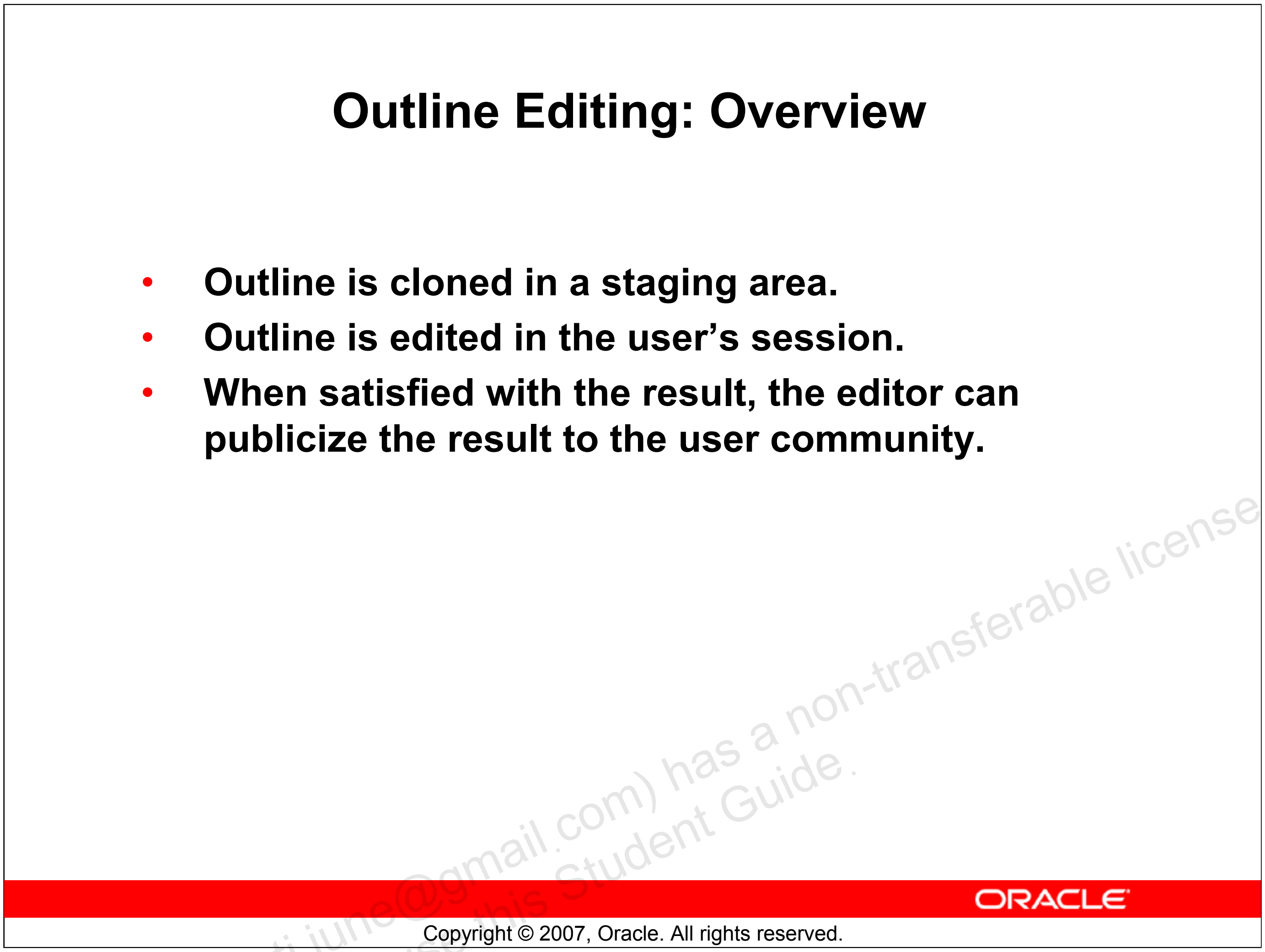
ORACLE

### Outline Editing: Overview

In the Oracle Database, outline editing extends the usefulness of stored outlines by introducing a user-editing interface. This interface enables users and third-party vendors to tune their execution plans by editing the stored outlines that are used to influence the optimizer. This can be useful when using the same application in different environments (small versus large databases).

Outline editing benefits both application developers and customer support personnel. Although the optimizer usually chooses optimal plans for queries, there are times when the user knows something about the execution environment that is inconsistent with the heuristics that the optimizer follows. Sometimes an execution plan is acceptable in one environment but not in another. By editing the outline directly, the user can tune the query without having to change the application.

The application developer might generate outlines in a staging area and notice that some plan did not take advantage of an index that could improve performance. It might be easier to edit the outline to use the index rather than searching through the application code and tuning the SQL until it eventually yields the desired result.



## Outline Editing: Overview

- Outline is cloned in a staging area.
- Outline is edited in the user’s session.
- When satisfied with the result, the editor can publicize the result to the user community.

### Outline Editing: Overview (continued)

For the customer whose environment has unique characteristics that might cause an outline to yield a less-than-optimal execution plan, the ability to make minor adjustments to the outline enhances the ability to support specific customer needs. In this sense, stored outlines are made more adaptive because users can make finely tuned adjustments to the saved plan.

Stored outline metadata is maintained in the `OUTLN` schema and maintained by the server. You are advised not to update these tables directly, just as you are advised not to update system tables. Therefore, the Oracle Database must provide users with a way to safely edit an outline without compromising the integrity of the outline for the rest of the user community.

To accomplish this, the Oracle Database proposes that the outline be cloned into the user’s schema at the onset of the outline editing session. All subsequent editing operations are performed on that clone until the user is satisfied with the edits and chooses to publicize them. In this way, any editing done by the user does not impact the rest of the user community, which continues to use the public version of the outline until the edits are explicitly saved.

# Editable Attributes

- Join order
- Join methods
- Access methods
- Distributed execution plans
- Distribution methods for parallel query execution
- Query rewrite
- View and subquery merging

## Editable Attributes

**Join order:** The join order defines the sequence in which tables are joined during query execution. This includes tables that are produced by evaluating subqueries and views as well as tables appearing in the `FROM` clauses of subqueries and views.

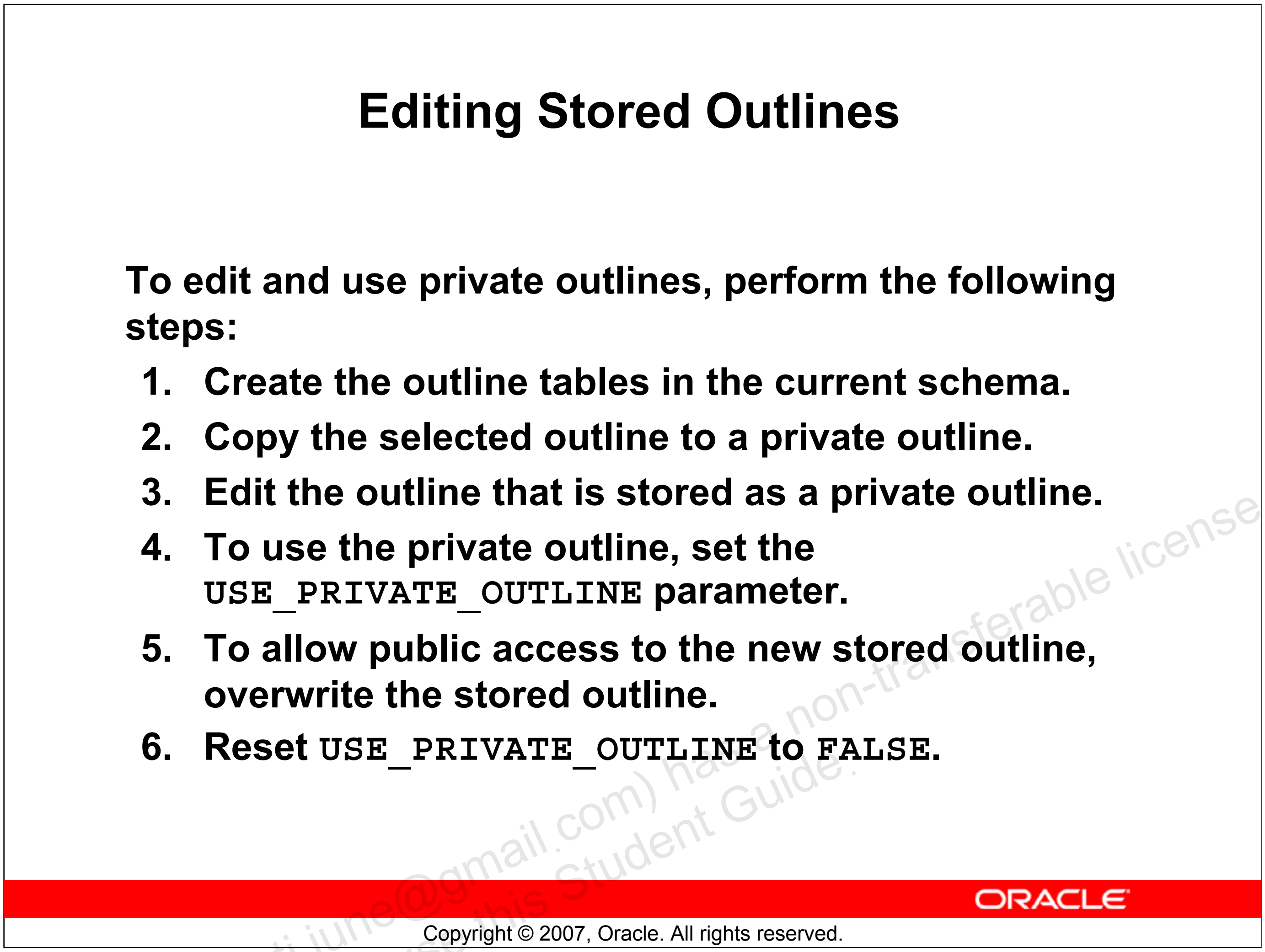
**Join methods:** Join methods define the methods that are used to join tables during query execution. Examples are nested loop joins and sort-merge joins.

**Access methods:** Access methods define the methods that are used to retrieve table data from the database. Examples are indexed access and full table scan.

**Distributed execution plans:** Distributed queries have execution plans that are generated for each site at which some portion of the query is executed. The execution plan for the local site at which the query is submitted can be controlled by plan stability, and equivalent plans must be produced at that site. In addition, driving site selection can be controlled centrally even though it might normally change after certain schema changes.

**Distribution methods:** For parallel query execution, distribution methods define how the inputs to execution nodes are partitioned.

**View and subquery merging and summary rewrite:** View and subquery merging and summary rewrite are meant to include all transformations in which objects or operations that occur in one subquery of the original SQL statement are caused to migrate to a different subquery for execution. Summary rewrite can also cause one set of objects or operations to be replaced by another.



# Editing Stored Outlines

To edit and use private outlines, perform the following steps:

1. Create the outline tables in the current schema.
2. Copy the selected outline to a private outline.
3. Edit the outline that is stored as a private outline.
4. To use the private outline, set the `USE_PRIVATE_OUTLINE` parameter.
5. To allow public access to the new stored outline, overwrite the stored outline.
6. Reset `USE_PRIVATE_OUTLINE` to `FALSE`.

### Editing Stored Outlines

Suppose that you want to edit the `DEV01` outline. The steps are as follows:

1. Connect to a schema from which the outlined statement can be executed, and make sure that the `CREATE ANY OUTLINE` and `SELECT` privileges have been granted. Use the `dbms_outln_edit.create_edit_tables` procedure to create outline editing tables locally.
2. Clone the outline being edited to the private area by using the following:

```
SQL> CREATE PRIVATE OUTLINE p_DEV01
  2  FROM dev01;
```

3. Edit the outline, either with the Outline Editor in Oracle Enterprise Manager or manually by querying the local `ol$hints` tables and performing DML against the appropriate hint tables. Use the `dbms_outln_edit.change_join_pos` procedure to change the join order.
4. If you are manually editing the outline, resynchronize the stored outline definition by using the “identity statement” as follows:

```
SQL> CREATE PRIVATE OUTLINE p_dev01
  2  FROM PRIVATE p_dev01;
```

### Editing Stored Outlines (continued)

To accomplish the same task as in step 4, you can also use:

```
SQL> dbms_outln_edit.refresh_private_outline;
```

or

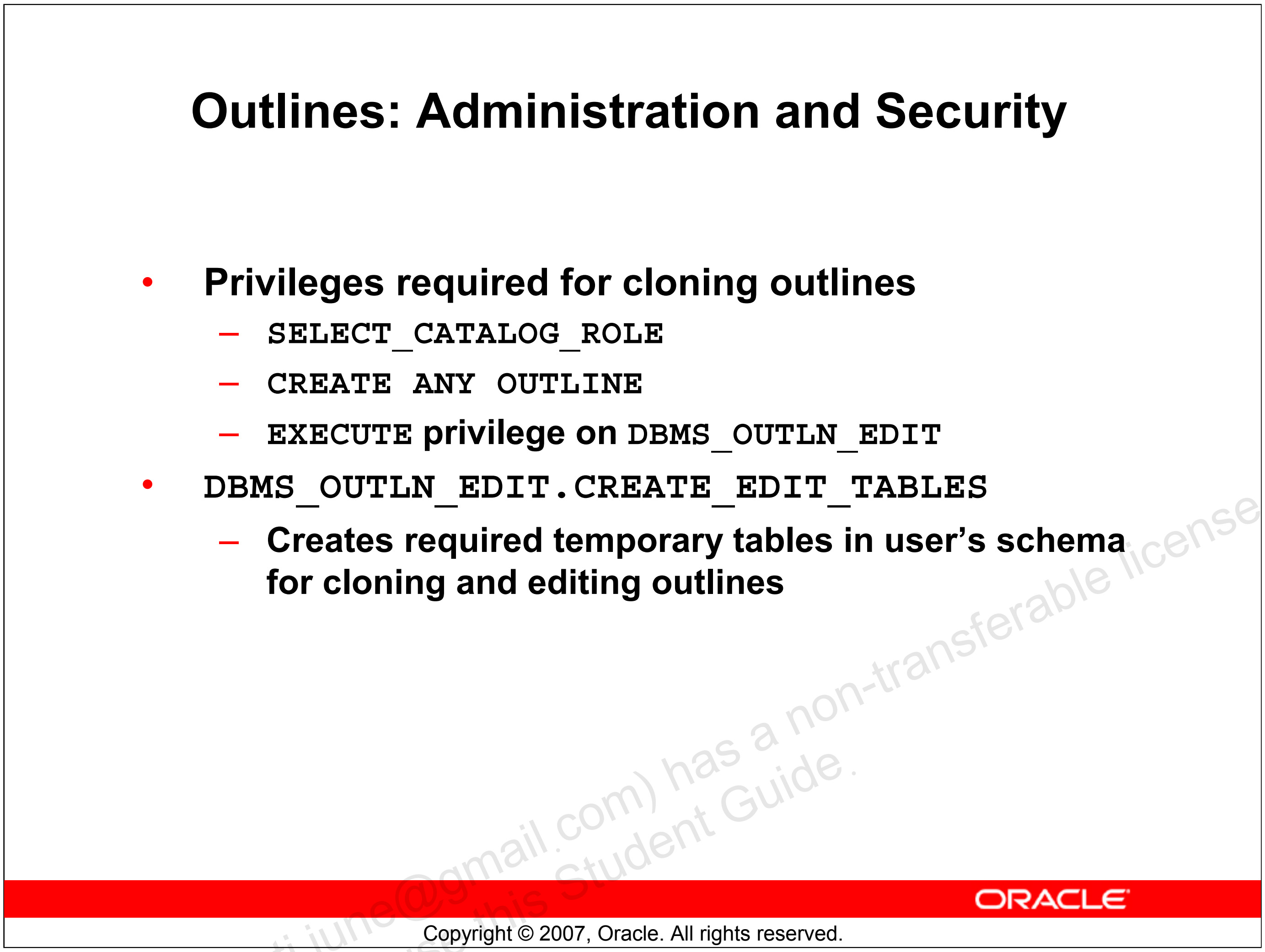
```
SQL> ALTER SYSTEM FLUSH SHARED_POOL;
```

Test the edits. Set USE_PRIVATE_OUTLINES to TRUE, and then issue the outline statement or run EXPLAIN PLAN on the statement.

5. If you want to preserve these edits for public use, publicize the edits with the following statement:

```
SQL> CREATE OR REPLACE OUTLINE dev01
  2  FROM PRIVATE p_dev01;
```

6. Disable private outline usage by setting the USE_PRIVATE_OUTLINES parameter to FALSE.



## Outlines: Administration and Security

- Privileges required for cloning outlines
  - `SELECT_CATALOG_ROLE`
  - `CREATE ANY OUTLINE`
  - `EXECUTE` privilege on `DBMS_OUTLN_EDIT`
- `DBMS_OUTLN_EDIT.CREATE_EDIT_TABLES`
  - Creates required temporary tables in user's schema for cloning and editing outlines

### Outlines: Administration and Security

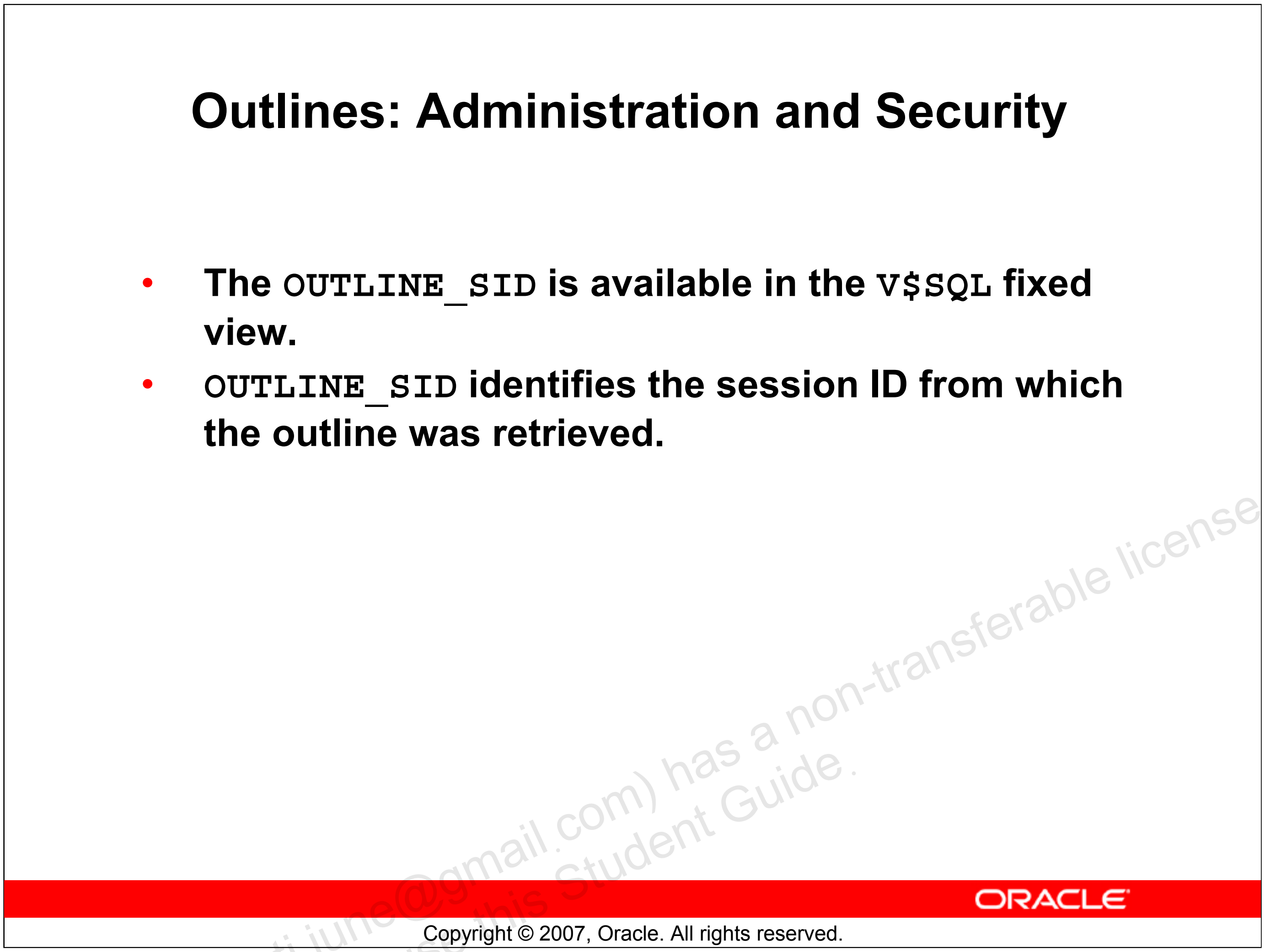
**`SELECT_CATALOG_ROLE`**

This role is required for the `CREATE OUTLINE FROM` command unless the issuer of the command is also the owner of the outline. Any `CREATE OUTLINE` statement requires the `CREATE ANY OUTLINE` privilege. Specifying the `FROM` clause requires the additional `SELECT_CATALOG_ROLE` role because such a command exposes SQL text to different users who may otherwise not be privileged to read the text.

**`DBMS_OUTLN_EDIT.CREATE_EDIT_TABLES`**

This is a supporting command procedure that creates the metadata tables in the invoker's schema. This procedure is callable by anyone with the `EXECUTE` privilege on the `DBMS_OUTLN_EDIT` package. (Refer to *Supplied PL/SQL Packages Reference* for more information.) Note also that the `DBMS_OUTLN` package is synonymous with `OUTLN_PKG`.

Private outlines are private to the session editing them. To accommodate the possibility of multiple users editing the same outline at the same time (which is not a good practice but is possible), the data is partitioned by session. This is really scratch data that is not intended for long-term use. Partitioning the data removes the need for users to worry about cleaning up when they are done because the temporary data goes away when the session is closed.



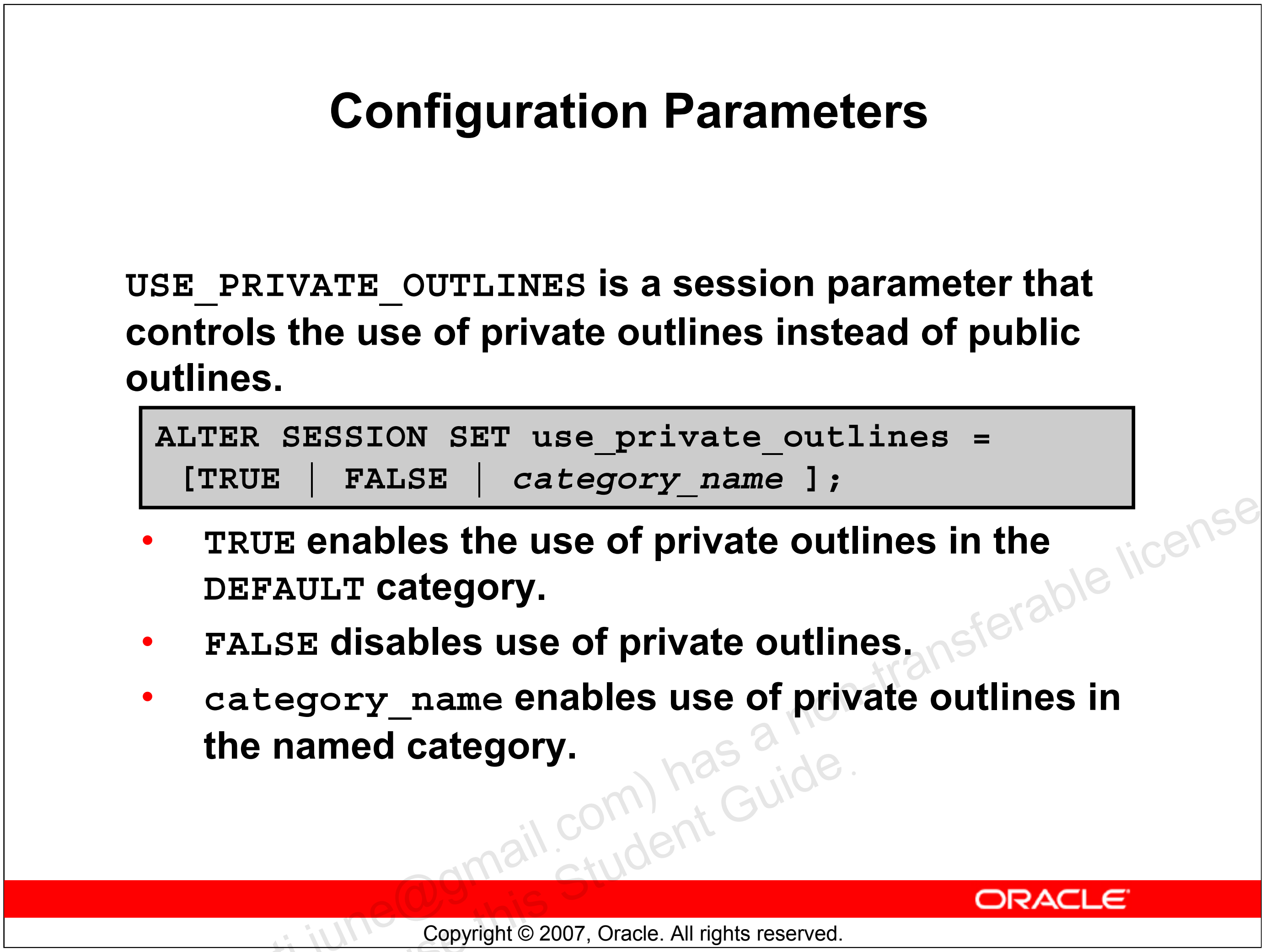
## Outlines: Administration and Security

- The `OUTLINE_SID` is available in the `V$SQL` fixed view.
- `OUTLINE_SID` identifies the session ID from which the outline was retrieved.

### Outlines: Administration and Security (continued)

**V$SQL**

A column is added to the `V$SQL` fixed view to help users determine whether a shared cursor was compiled while using a private outline or a public outline. `OUTLINE_SID` is the name of this new column, and it identifies the session ID from which the outline was retrieved. The default value is `0`, which implies a lookup in the `OUTLN` schema.



## Configuration Parameters

USE_PRIVATE_OUTLINES is a session parameter that controls the use of private outlines instead of public outlines.

```
ALTER SESSION SET use_private_outlines =
  [TRUE | FALSE | category_name ];
```

- TRUE enables the use of private outlines in the DEFAULT category.
- FALSE disables use of private outlines.
- *category_name* enables use of private outlines in the named category.

### Configuration Parameters

The USE_PRIVATE_OUTLINES session parameter is added to control the use of private outlines instead of public outlines. When an outlined SQL command is issued, this parameter causes outline retrieval to come from the session private area rather than from the public area that is usually consulted according to the setting of USE_STORED_OUTLINES. If no outline exists in the session private area, no outline is used for the compilation of the command.

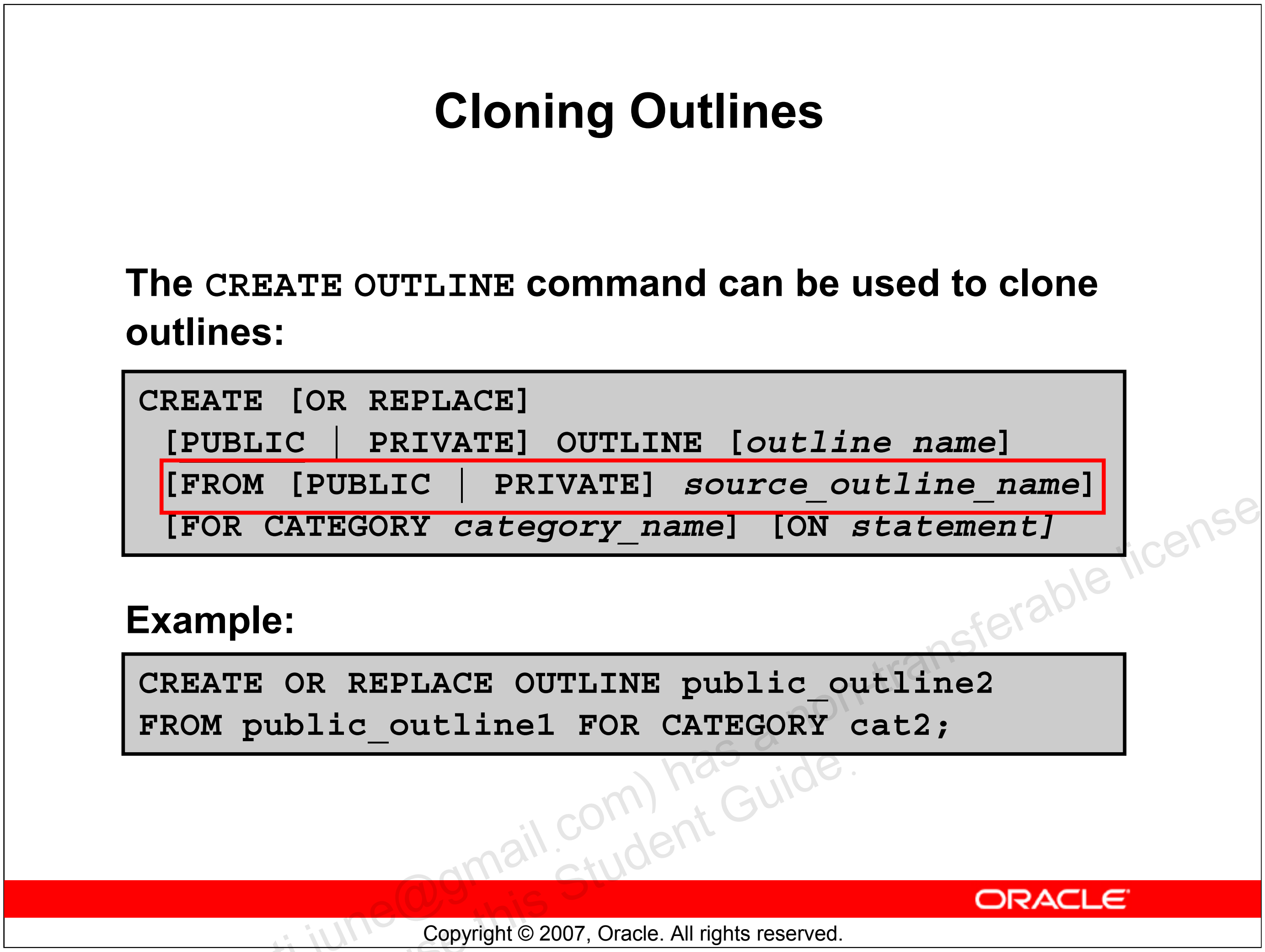
You can specify a value for this session parameter by using the following syntax:

```
ALTER SESSION SET USE_PRIVATE_OUTLINES =
   TRUE | FALSE | category_name ;
```

Here are details about the syntax:

- TRUE enables use of private outlines and defaults to the DEFAULT category.
- FALSE disables use of private outlines.
- *category_name* enables use of private outlines in the named category.

When a user begins an outline-editing session, the parameter should be set to the category to which the outline being edited belongs. This enables the feedback mechanism because it allows the private outline to be applied to the compilation process. On completion of outline editing, this parameter should be set to FALSE to restore the session to normal outline lookup as dictated by the USE_STORED_OUTLINES parameter.



# Cloning Outlines

**The CREATE OUTLINE command can be used to clone outlines:**

```
CREATE [OR REPLACE]
  [PUBLIC | PRIVATE] OUTLINE [outline name]
  [FROM [PUBLIC | PRIVATE] source_outline_name]
  [FOR CATEGORY category_name] [ON statement]
```

**Example:**

```
CREATE OR REPLACE OUTLINE public_outline2
FROM public_outline1 FOR CATEGORY cat2;
```

## Cloning Outlines

Here are some details about the cloning syntax in the slide.

PUBLIC: The outline is to be created for use by PUBLIC. This is the default because outline creation is intended for systemwide use.

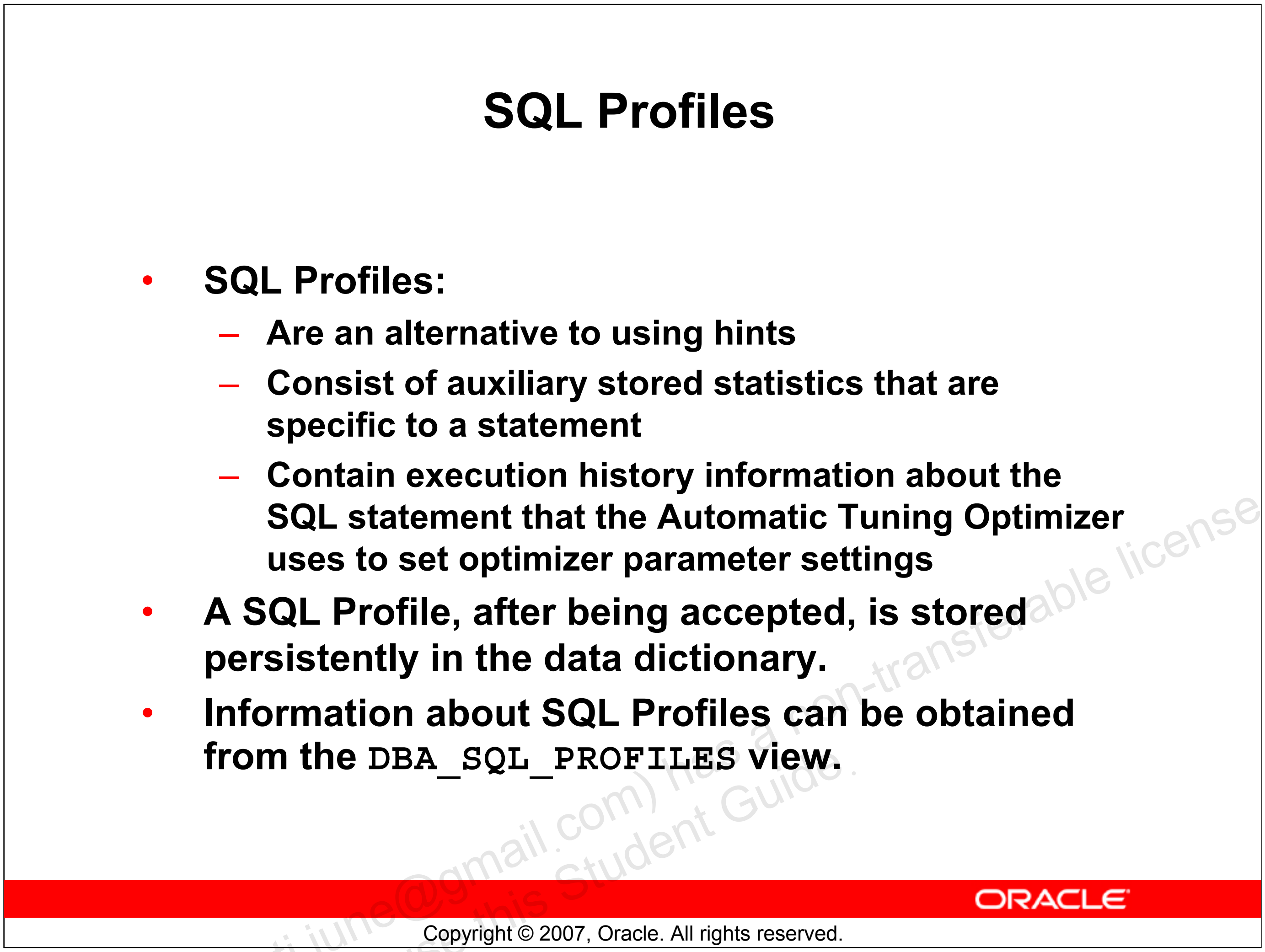
PRIVATE: The outline is to be created for private use by the current session only, and its data is stored in the current parsing schema. When specified, the prerequisite outline tables and indexes must exist in the local schema.

FROM: This construct provides a way to create an outline by copying an existing one.

*source_outline_name*: This is the name of the outline being cloned. By default, it is found in the public area. However, if it is preceded by the PRIVATE keyword, it is found in the local schema.

The addition of the PRIVATE and FROM keywords enables outline cloning. When you want to edit an outline, you do so on a private copy that is created by specifying the PRIVATE keyword. In the FROM clause, the source outline to be edited is named and is found in the public area unless preceded by the PRIVATE keyword, in which case you would be copying a private version of the named outline.

When specifying the FROM clause, existing semantics apply to the outline name and category. If unspecified, therefore, an outline name is generated under the DEFAULT category. When a PRIVATE outline is being created, an error is returned if the prerequisite outline tables that hold the outline data do not exist in the local schema.



## SQL Profiles

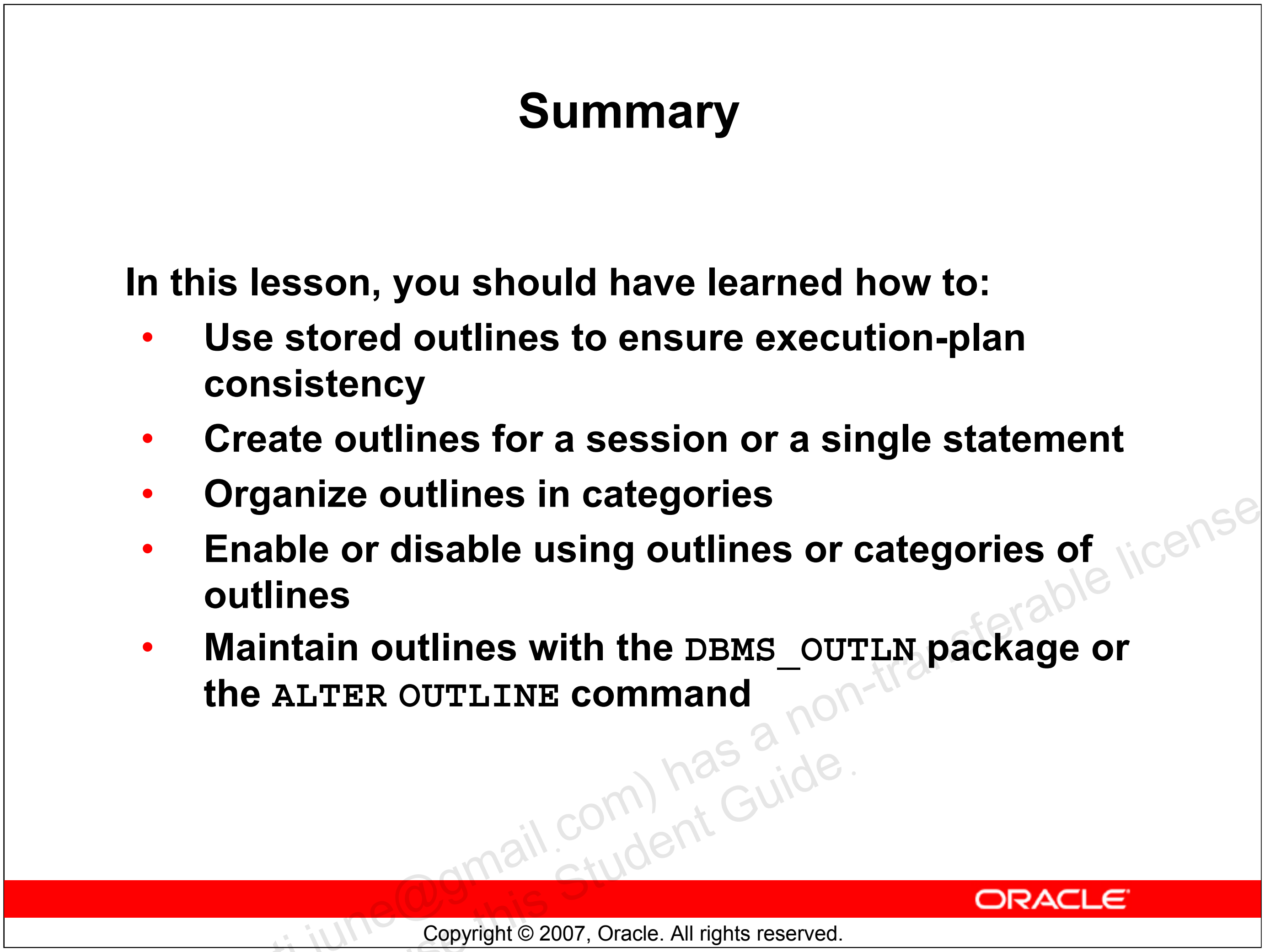
- SQL Profiles:
  - Are an alternative to using hints
  - Consist of auxiliary stored statistics that are specific to a statement
  - Contain execution history information about the SQL statement that the Automatic Tuning Optimizer uses to set optimizer parameter settings
- A SQL Profile, after being accepted, is stored persistently in the data dictionary.
- Information about SQL Profiles can be obtained from the `DBA_SQL_PROFILES` view.

### SQL Profiles

The Automatic Tuning Optimizer creates a SQL Profile, which is a SQL statement profile that consists of auxiliary statistics that are specific to that statement. The query optimizer under normal mode makes estimates about cardinality, selectivity, and cost that can sometimes be wrong by a significant amount, resulting in poor execution plans. A SQL Profile addresses this problem by collecting additional information using sampling and partial execution techniques to verify and (if needed) adjust the estimates.

During SQL profiling, the Automatic Tuning Optimizer also uses execution history information about the SQL statement to appropriately set optimizer parameter settings, such as changing the `OPTIMIZER_MODE` initialization parameter setting from `ALL_ROWS` to `FIRST_ROWS` for that SQL statement. The output of this type of analysis is a recommendation to accept the SQL Profile. A SQL Profile, once accepted, is stored persistently in the data dictionary. Note that the SQL Profile is specific to a particular query.

If accepted, the optimizer under normal mode uses the information in the SQL Profile in conjunction with regular database statistics when generating an execution plan. The availability of the additional information makes it possible to produce well-tuned plans for a corresponding SQL statement without requiring changes to the application code. Information about SQL Profiles can be obtained in the `DBA_SQL_PROFILES` view.



## Summary

In this lesson, you should have learned how to:

- Use stored outlines to ensure execution-plan consistency
- Create outlines for a session or a single statement
- Organize outlines in categories
- Enable or disable using outlines or categories of outlines
- Maintain outlines with the `DBMS_OUTLN` package or the `ALTER OUTLINE` command

ORACLE

### Summary

This lesson introduced you to stored outlines and their influence on optimizer plan stability. The lesson also introduced you to using the DBMS_OUTLN package on stored outlines.

The Oracle Database provides you with a means of stabilizing execution plans across Oracle Database releases, database changes, or other factors that can cause an execution plan to change. You can create a stored outline containing a set of hints that are used by the optimizer to create an execution plan.

Stored outlines rely partially on hints that the optimizer uses to achieve stable execution plans. These plans are maintained through many types of database and instance changes. You can use stored outlines to ensure that all your customers access the same execution plans.

You can create outlines for a session, or you can create them for specific SQL statements.

To use stored outlines, the USE_STORED_OUTLINES option must be set to TRUE or to a category name. Use procedures in the DBMS_OUTLN package to manage stored outlines and their categories. There are data dictionary views that store information on the outlines.