

---

# Appendix A

## Practices and Solutions

---

## Table of Contents

Practices for Lesson 1 .....	4
Practices for Lesson 2 .....	5
Practice 2-1: Viewing Diagnostic Information .....	6
Practice 2-2: Using the Alert Log .....	10
Practice 2-3: Viewing System Statistics and Wait Events .....	13
Practices for Lesson 3 .....	18
Practice 3-1: Using Automatic Workload Repository .....	19
Practices for Lesson 4 .....	29
Practice 4-1: Using Enterprise Manager to Identify OS Issues .....	30
Practices for Lesson 5 .....	32
Practice 5-1: Using Metrics .....	33
Practices for Lesson 6 .....	43
Practice 6-1: Using Baselines .....	44
Practices for Lesson 7 .....	51
Practice 7-1: Using AWR-Based Tools .....	52
Practices for Lesson 8 .....	63
Practice 8-1: Using Services with a Single-Instance Oracle Database .....	64
Practice 8-2: Tracing Services in a Single-Instance Oracle Environment .....	76
Practices for Lesson 9 .....	83
Practice 9-1: Using AUTOTRACE and DBMS_XPLAN .....	84
Practice 9-2: Using the SQL TRACE Facility .....	89
Practices for Lesson 10 .....	93
Practice 10-1: Capturing Extended Statistics .....	94
Practice 10-2: Stale Statistics .....	99
Practices for Lesson 11 .....	104
Practice 11-1: Excess Blocks .....	105
Practice 11-2: Coalescing an Index .....	110
Practices for Lesson 12 .....	117
Practice 12-1: Using the SQL Performance Analyzer .....	118
Practices for Lesson 13 .....	128
Practice 13-1: Seeding SQL Plan Baselines from SQL Performance Advisor .....	129
Practice 13-2: SQL Plan Management (SPM) .....	136
Practices for Lesson 14 .....	166
Practice 14-1: Using Database Replay .....	167
Practices for Lesson 15 .....	176
Practice 15-1: Sizing the Shared Pool .....	177
Practice 15-2: Tuning a Hard-Parse Workload .....	189
Practice 15-3: Keeping Objects in the Shared Pool .....	198
Practices for Lesson 16 .....	201
Practice 16-1: Using the DB Cache Advisor .....	202
Practice 16-2: Using the Keep Pool .....	208
Practices for Lesson 17 .....	216
Practice 17-1: Tuning PGA_AGGREGATE_TARGET .....	217
Practices for Lesson 18 .....	229

Practice 18-1: Enabling Automatic Shared Memory .....	230
Practice 18-2: Adjusting Memory as Workloads Change.....	237
Practices for Lesson 19 .....	241
Practices for Lesson 20 .....	242
Practices for Appendix B .....	243
Practice for Appendix B-1: Installing Statspack.....	244
Practice for Appendix B-2: Creating Snapshots .....	245
Practice for Appendix B-3: Generating Statspack Reports.....	248

## Practices for Lesson 1

There are no practices for this lesson.

## Practices for Lesson 2

In this practice, you look at various diagnostics to view the types of information available in the database.

## Practice 2-1: Viewing Diagnostic Information

In this practice, you use the alert log to gather information about the performance of the instance while running a workload. You review the alert log for information that is relevant to performance tuning.

1. Set up the `orcl` database for a workload. Execute the prepare script:

```
./prepare 2 1
```

```
$ cd /home/oracle/workshops
$ ./prepare 2 1

Database closed.
Database dismounted.
ORACLE instance shut down.
ORACLE instance started.

Total System Global Area  836976640 bytes
Fixed Size                  1339740 bytes
Variable Size               805310116 bytes
Database Buffers            25165824 bytes
Redo Buffers                 5160960 bytes
Database mounted.
Database opened.
```

2. Run a workload by using the workload generator: `./workgen 2 1`.  
Check whether the workload has started by using the `ps` command in the same terminal window. You see a set of processes as illustrated in the following screenshot. The order and number of each vary. In most cases, you see several `.sql` scripts, `sqlplus` and `sleep` processes. Not all these processes may be active at the time that the `ps` command is issued. This workload continues until it is stopped. Continue to the next step in the practice.

```
$ cd /home/oracle/workshops
$ ./workgen 2 1

PL/SQL procedure successfully completed.

PL/SQL procedure successfully completed.

$ ps
  PID TTY          TIME CMD
 25946 pts/2    00:00:00 bash
 30124 pts/2    00:00:00 insert_orders.s
 30125 pts/2    00:00:00 update_orders.s
 30126 pts/2    00:00:00 delete_orders.s
 30130 pts/2    00:00:00 insert_orders.s
 30131 pts/2    00:00:00 update_orders.s
 30134 pts/2    00:00:00 delete_orders.s
 30135 pts/2    00:00:00 insert_orders.s
```

## Practice 2-1: Viewing Diagnostic Information (continued)

```
...  
$
```

3. Start the Enterprise Manager Database Control by starting the browser with the URL: <https://<hostname>:1158/em>.

The <hostname> is the name of your PC. You can find that name with the hostname command.

```
$ hostname
```

Log in as the SYS user with the password, oracle\_4U, and connect as SYSDBA.

**Note:** Passwords are case-sensitive by default in Oracle Database 11g.

On the Database home page in Enterprise Manager, set View Data to Automatically (15 Sec). On the Database home page, what diagnostic information is available?

\_\_\_\_\_ (Host CPU and Active Sessions graphs are available after a few refreshes.)

4. Observe the characteristics of the workload on the server. This is shown in the Host CPU graph.
  - a) What is the load shown in the Host CPU graph? \_\_\_\_\_  
(This number varies at each refresh.)
  - b) What percentage of the CPU is the orcl processes using? \_\_\_\_\_
5. The characteristics of the workload on the database instance are shown in the Active Sessions graph. When there are a large number of waiting sessions relative to the number of sessions that use the CPU, you must check other diagnostics.
  - a) What is the average number of sessions that use the CPU?  
\_\_\_\_\_
  - b) What is the average number of sessions that are waiting? \_\_\_\_\_, number waiting on I/O \_\_\_\_\_
6. How many CPU cores are in your system? \_\_\_\_\_

The Core Count below the Active Session Graph shows the number of CPU cores in your system. Another way to determine the number of CPUs in your system: Click the Database tab. Click the Load link in the Host CPU section. Click CPU Usage in the CPU Utilization section. The CPU Usage page shows one row for each CPU in your system.

7. Navigate to the Performance page. Click the Database tab, and then the Performance tab at the top of the page.
8. On the Performance page, load average (Average Runnable Processes) and average active sessions are graphed. The number of CPU cores can be shown in the graph by selecting the Show CPU Cores check box.

## Practice 2-1: Viewing Diagnostic Information (continued)

- a) What is the value of the load average in relation to the maximum CPU?  
 \_\_\_\_\_ (The load average is greater than the maximum CPU.) When the load average is greater than the maximum CPU, processes need to wait on the run queue.
9. In the Average Active Sessions graph, the various colors of the graph show various components of the waits and service times that the sessions are experiencing. Each component shows the number of active sessions in that state. For example, the component at the bottom of the graph shows the number of active sessions that use the CPU.
  - a) Which component of the graph, other than the CPU, consistently has the highest number of active sessions? \_\_\_\_\_ (This can vary. CPU Wait, Concurrency, Configuration, User I/O, and Commit are valid answers.)
10. Click Concurrency in the Average Active Sessions graph legend to view the waits that are included in this category. From a visual scan of the Active Sessions Waiting: Concurrency page, what are the highest waits?  
 \_\_\_\_\_ (Buffer busy waits, Enq TX: index contention, Library Cache Mutex X, and Latch: Cache Buffer chains are all valid answers.) **Note:** In 11g Release 2, the internal algorithms have changed making buffer busy waits less likely.
11. Navigate to the Server page. Click the Database tab at the top of the page, and then click the Server tab.
12. On the Server page, click Automatic Workload Repository in the Statistics Management section.
13. On the Automatic Workload Repository page, click the number beside Snapshots in the Manage Snapshots and Baselines section.
14. On the Snapshots page, note the last snapshot number shown: \_\_\_\_\_. (This number may be over 100.)
  - a) Create a new snapshot. Click Create.
  - b) On the Confirmation page, click Yes.
15. Create an AWR report.
  - a) On the Snapshots page, select the snapshot that you recorded in step 14. Select View Report from the Actions drop-down menu, and then click Go beside the Action menu.
  - b) On the View Report page, select the next snapshot after the one chosen in step 14 a. Click OK.
16. When the report appears, scroll down to Top 5 Timed Foreground Wait Events.
  - a) What is the top timed event? \_\_\_\_\_
  - b) What is the second timed event? \_\_\_\_\_



## **Practice 2-1: Viewing Diagnostic Information (continued)**

- c) Are any of the other timed events found in step 10 listed here? If so, list the timed events that are found.

---

17. Scroll down to the Wait Events Statistics section. Click the Time Model Statistics link.

- a) Which category collects the most %DB Time? \_\_\_\_\_  
(sql execute elapsed time is expected but this item might be second.)
- b) Can you guess what performance issue is slowing this instance?  
\_\_\_\_\_ (Poor SQL execution plans that produce index contention may be contributing to the sql elapsed time. In addition, log file waits may be holding up commit processing also contributing to SQL elapsed time)
- c) Stop the workload. In the terminal window, issue the following command:  

```
$ rm $HOME/workshops/runload
```

**NOTE:** Most workload scripts in this course run until stopped. They repeat as long as the \$HOME/workshops/runload file exists.

18. Click the Database tab at the top of the page to return to the Database Instance home page.

## Practice 2-2: Using the Alert Log

In this practice, you use the alert log to gather information about instance configuration and performance.

1. View the alert log through Enterprise Manager (EM) Database Control. Find the last 500 entries.
  - a) Invoke the browser (Mozilla) and enter the URL for Enterprise Manager Database Control for the `orcl` database. Use the following URL:  
`https://<hostname>:1158/em`
  - b) Log in to EM for the `orcl` database.
  - c) From the Database home page, navigate to the Alert Log Contents page. Scroll to the bottom of the Database home page and click the Alert Log Contents link in the Related Links section.
  - d) In the View Entries box, select Last 500 and click Go.
  - e) Scroll through the entries. Note that the entries are in reverse chronological order. The most recent entries are at the top of the page.
2. Find the current Redo Log number and how many redo logs have been generated since yesterday. Each time the log file is full, a log switch occurs and a message is sent to the alert log giving the current log number.
  - a) Click Search. On the Search Alert Log Contents page, enter Most Recent 25 Hours. In the Message Text box, enter `. *Current log. *` and select the Regular Expression check box. Then click Go.  
 Current sequence number (seq#): \_\_\_\_\_ (140, varies)  
 Number of log switches: \_\_\_\_\_
  - b) It is recommended that log switches take place no more than every 20 minutes. Is the number of log switches excessive? \_\_\_\_\_  
 (Yes. To reduce the number of log switches, increase the size of the redo log files.)
3. Find the name of the current SPFILE and the names of any other SPFILES that have been used in the last 10 days. Use the Search feature of the View Alert Log Contents page.
  - a) Use the Search feature again. Set the date range to Most Recent 10 Days, and then change the message text to `. *spfile. *`. Ensure that the Regular Expression check box is selected.  
 \_\_\_\_\_  
 (/u01/app/oracle/product/11.2.0/dbhome\_1/dbs/spfileo  
 rcl.ora)

## Practice 2-2: Using the Alert Log (continued)

- b) Why is the SPFILE that is used to start the database important?

\_\_\_\_\_  
(Changes in the initialization parameters can have a large effect on database performance.)

- c) How else can initialization parameters change?  
\_\_\_\_\_(by the use of ALTER SYSTEM commands)
- d) Have any initialization parameters been changed using the ALTER SYSTEM command? (Hint: Use . \*ALTER SYSTEM. \* in the message text field.)

Note: The prepare script issues ALTER SYSTEM commands.

4. From the command line, find and view the alert log file for the orcl database. Use the operating system login, and locate and view the log for the orcl database. The text version of the alert log in Oracle Database 11g is located in the directory specified by the Diag Trace entry in the V\$DIAG\_INFO view.
- a) Open a terminal window. Start sqlplus and connect as sysdba. Execute the following command to find the Diag Trace directory. Exit sqlplus.

```
$ sqlplus / as sysdba

/* version and option banner omitted */

SQL> select * from V$DIAG_INFO
  2  where name = 'Diag Trace';

INST_ID NAME
-----
VALUE
-----
          1 Diag Trace
/u01/app/oracle/diag/rdbms/orcl/orcl/trace

SQL> exit
```

- b) In the terminal window, change directories to the value of the Diag Trace directory. Then, view the alert\_orcl.log file. (gedit, a WYSIWYG editor, and text file viewers—less and more—are available.)
- c) Find the time of the last startup. Start at the end of the file. (Search from the most recent entries to the “Starting ORACLE instance” string.) The time stamp for this event is the time stamp immediately above the entry.

Time of the last startup \_\_\_\_\_

5. Determine how often the log files have switched in the last 10 minutes, while the workload is running.

## Practice 2-2: Using the Alert Log (continued)

- a) The log file may have switched several times. The `tail` command lists the last 10 lines of a file. The `-n 100` option causes the `tail` command to display 100 lines; you may need to use `tail` with more lines to see a full 10 minutes.
6. While viewing the alert log, find the settings for the `db_cache_size`, `log_buffer`, `sga_max_size`, and `compatible` parameters.  
**Hint:** All nondefault parameter values are listed in the alert log on every instance startup.
- a) Scroll through the alert log entries. Start from the most recent entries in the file to find the current parameter setting. Note that with this method, you see only those parameters that have been set to a nondefault value.  
 (log\_buffer is not visible because it has a default value.)
7. Use SQL\*Plus to find the values for the `db_cache_size`, `log_buffer`, `sga_max_size`, and `compatible` parameters. Use the `SHOW PARAMETERS` command.

```
$ sqlplus / as sysdba

/* version and option banner omitted */

SQL> show parameter db_cache_size
```

NAME	TYPE	VALUE
db_cache_size	big integer	24M

```
SQL> show parameter log_buffer
```

NAME	TYPE	VALUE
log_buffer	integer	4980736

```
SQL> show parameter sga_max_size
```

NAME	TYPE	VALUE
sga_max_size	big integer	800M

```
SQL> show parameter COMPATIBLE
```

NAME	TYPE	VALUE
compatible	string	11.2.0.0.0

```
SQL> exit
```

## Practice 2-3: Viewing System Statistics and Wait Events

In this practice, you use dynamic performance views to view tuning information.

1. Start the workload generator and continue. The workload continues until it is stopped.

```
$ cd /home/oracle/workshops
$ ./workgen 2 3

PL/SQL procedure successfully completed.

PL/SQL procedure successfully completed.
$
```

2. View the system statistics in SQL\*Plus as the SYS user. Use the V\$SYSSTAT view and select NAME, CLASS, and VALUE. Order the output by CLASS. These statistics are cumulative from the time that the instance was started.
  - a) What can you deduce about the instance workload and performance from a listing of the system statistics?

**Answer:** There is very little that you can determine from a listing of the system statistics by itself. Two listings that are used to create a difference in values over a known period of time can yield rate and load information.

```
$ sqlplus / as sysdba
SQL> COLUMN name FORMAT A40 WORD_WRAPPED

SQL> SELECT name, class, value
2 FROM V$SYSSTAT
3 ORDER BY class;
```

3. View system wait events. Use the V\$SYSTEM\_EVENT view to see wait event statistics. Find the total number of waits, the time waited, and the average time waited for the log file sync and enq: TX - index contention events, and the time waited relative to the other wait events. The number of rows returned will vary; events that have zero counts will not be listed.

**Answer:** The log file sync and enq: TX - index contention events have collected some waits and the average wait time is displayed. There are several wait events that are not important, such as rdbms ipc message and smon timer. These are internal waits for a command to be issued, or for a timer to expire to initiate a task. They do not affect performance. Your output from this query will vary from the example shown.

```
SQL> COLUMN event FORMAT A30 WORD_WRAPPED
SQL> COLUMN average_wait HEADING AVERAGE|WAIT
SQL> COLUMN TIME_WAITED HEADING TIME|WAITED
SQL> COLUMN TOTAL_WAITS HEADING TOTAL|WAITS
SQL> SET PAGESIZE 60
```

## Practice 2-3: Viewing System Statistics and Wait Events (continued)

```
SQL> SELECT event, total_waits, time_waited, average_wait
2 FROM V$SYSTEM_EVENT
3 ORDER BY time_waited DESC;
```

EVENT	TOTAL WAITS	TIME WAITED	AVERAGE WAIT
-----	-----	-----	-----
rdbms ipc message	127563	1866712	14.63
SQL*Net message from client	66606	1351490	20.29
DIAG idle wait	3258	347413	106.63
jobq slave wait	3782	189368	50.07
pmon timer	999	173713	173.89
wait for unread message on broadcast channel	1735	172055	99.17
Streams AQ: qmn slave idle wait	63	171667	2724.88
shared server idle wait	58	171575	2958.19
Space Manager: slave idle wait	413	170408	412.61
Streams AQ: qmn coordinator idle wait	68	170226	2503.32
dispatcher timer	28	168073	6002.59
Streams AQ: waiting for messages in the queue	336	168025	500.07
smon timer	63	161556	2564.38
log file parallel write	331932	38612	.12
class slave wait	19	29965	1577.11
log file sync	7952	10467	1.32
Streams AQ: waiting for time management or cleanup tasks	4	6977	1744.37
db file sequential read	25281	5859	.23
db file async I/O submit	871	4727	5.43
Data file init write	440	1116	2.54
enq: HW - contention	26	1036	39.83
db file scattered read	2852	759	.27
buffer busy waits	63	751	11.92
os thread startup	99	583	5.89
Streams AQ: qmn coordinator waiting for slave to start	3	500	166.67
control file parallel write	1119	428	.38
control file heartbeat	1	400	399.99
library cache: mutex X	11	357	32.47
LGWR wait for redo copy	1223	275	.22
SQL*Net break/reset to client	138	217	1.58
db file parallel read	88	209	2.37
latch: cache buffers chains	25	182	7.27

## Practice 2-3: Viewing System Statistics and Wait Events (continued)

buffer deadlock	2	153	76.59
cursor: pin S	2	144	72.06
log file switch completion	5	142	28.4
Disk file operations I/O	9510	129	.01
latch free	105	129	1.22
JS coord start wait	2	100	50.07
ADR block file read	135	79	.58
latch: redo allocation	1126	75	.07
rdbms ipc reply	42	69	1.64
latch: In memory undo latch	22	56	2.53
latch: redo copy	1	56	56.03
latch: cache buffers lru chain	9	45	4.96
EVENT	TOTAL WAITS	TIME WAITED	AVERAGE WAIT
-----	-----	-----	-----
asynch descriptor resize	6982	44	.01
direct path read	398	40	.1
log file single write	56	40	.71
enq: TX - row lock contention	11	36	3.25
latch: shared pool	20	27	1.36
enq: TX - allocate ITL entry	1	23	23.38
db file single write	242	19	.08
latch: call allocation	4	16	3.93
SQL*Net message to client	62550	15	0
latch: enqueue hash chains	11	14	1.29
control file sequential read	8236	14	0
local write wait	6	11	1.81
ADR block file write	6	11	1.78
row cache lock	1	9	8.52
direct path sync	5	7	1.44
enq: TX - index contention	1	7	6.8
latch: messages	150	6	.04
enq: CN - race with init	1	4	4.12
read by other session	2	3	1.34
log file switch (private strand flush incomplete)	3	2	.79
direct path write	56	2	.04
reliable message	6	1	.14
SQL*Net more data to client	165	1	.01
CRS call completion	1	1	1.17
log file sequential read	56	0	0
enq: KO - fast object checkpoint	1	0	.3
direct path write temp	1	0	.01
enq: US - contention	2	0	.03
latch: checkpoint queue latch	1	0	0
instance state change	1	0	.06
ADR file lock	8	0	0

## Practice 2-3: Viewing System Statistics and Wait Events (continued)

latch: session allocation	1	0	.04
SQL*Net more data from client	82	0	0
latch: row cache objects	3	0	.13

78 rows selected.

4. Check the session wait events. Find the sessions that are currently waiting on the log file sync event. Use the V\$SESSION\_WAIT view. Why are very few or no sessions listed?

**Answer:** The V\$SESSION\_WAIT view shows waits only for currently connected sessions. The waits for all sessions, past and current, are summed in the V\$SYSTEM\_EVENTS view. Your output from this query will vary from the example shown.

```
SQL> COLUMN time_waited HEADING TIME|WAITED
SQL> COLUMN AVERAGE_WAIT HEADING AVERAGE|WAIT
SQL> COLUMN seconds_in_wait HEADING SECONDS|IN_WAIT
SQL> COLUMN EVENT FORMAT A20
SQL> COLUMN total_waits HEADING TOTAL|WAITS
SQL>
SQL> select sid, wait_time, seconds_in_wait, state
2      from v$session_wait
3      where event = 'log file sync';
```

no rows selected

```
SQL>
SQL>
SQL> SELECT sid, event, total_waits,
2      time_waited, average_wait
3      FROM V$SESSION_EVENT
4      WHERE event = 'log file sync';
```

SID	EVENT	TOTAL WAITS	TIME WAITED	AVERAGE WAIT
1	log file sync	1	0	.11
20	log file sync	262	106	.4
25	log file sync	3	7	2.37
26	log file sync	24	13	.56
32	log file sync	4	2	.39
33	log file sync	160	114	.71
36	log file sync	1	0	.03
37	log file sync	1	0	.07
38	log file sync	1	0	.05

9 rows selected.

SQL> exit



### ***Practice 2-3: Viewing System Statistics and Wait Events (continued)***

5. Stop the workload by executing the `rm runload` command. Then, clean up the practice environment by running the `./cleanup 2 3` script. (Cleanup may not be needed.)

```
$ cd /home/oracle/workshops
$ rm /home/oracle/workshops/runload
$ ./cleanup 2 3
System altered.
$
```

## Practices for Lesson 3

During this practice, you use AWR snapshots and reports to identify and fix the issues that you discover in a running workload.

### ***Practice 3-1: Using Automatic Workload Repository***

In this practice, you run a workload on the `orcl` database, capture AWR snapshots, and diagnose and fix a performance issue. Note that during this practice, you explicitly capture AWR snapshots. However, by default, the system automatically captures them every hour.

1. Execute the `./prepare 3 1` script to set up this practice. You find this script in your `$HOME/workshops` directory. Log out of Enterprise Manager.

```
$ cd $HOME/workshops
$ ./prepare 3 1

drop tablespace tbsspc including contents and datafiles
*
ERROR at line 1:
ORA-00959: tablespace 'TBSSPC' does not exist

Tablespace created.

PL/SQL procedure successfully completed.

PL/SQL procedure successfully completed.

User dropped.

User created.

Grant succeeded.

drop table spct purge
*
ERROR at line 1:
ORA-00942: table or view does not exist

Table created.

PL/SQL procedure successfully completed.

Database closed.
Database dismounted.
ORACLE instance shut down.
```

## Practice 3-1: Using Automatic Workload Repository (continued)

```

ORACLE instance started.

Total System Global Area  836976640 bytes
Fixed Size                  1339740 bytes
Variable Size              805310116 bytes
Database Buffers           25165824 bytes
Redo Buffers                5160960 bytes
Database mounted.
Database opened.
$

```

2. Take your first AWR snapshot. You can use either Enterprise Manager or the DBMS\_WORKLOAD\_REPOSITORY package. The PL/SQL solution is shown in the sol\_03\_01\_02.sh script located in your \$HOME/solutions directory. The script creates an AWR snapshot by using the DBMS\_WORKLOAD\_REPOSITORY package. The solution in this document uses Enterprise Manager.
  - a) First, open your browser by using the URL:  
https://<hostname>:1158/em. This takes you to the EM Login page. Enter the credentials for the SYS user and click Login.
  - b) On the Database Instance page, click the Server tab.
  - c) On the Server page, click the Automatic Workload Repository link in the Statistics Management section.
  - d) On the Automatic Workload Repository page, click the number corresponding to the Snapshots field. The number of snapshots that you see will vary.

### Manage Snapshots and Baselines

Snapshots 2

Baselines 1

Latest Snapshot Time **Feb 13, 2008 10:23:23 PM**

Earliest Snapshot Time **Feb 13, 2008 11:30:07 AM**

- e) On the Snapshots page, click Create.
  - f) On the Confirmation page, click Yes.
  - g) The processing page appears. Wait until the snapshot is created.
  - h) The Confirmation box appears on the Snapshots page. You can see that a new snapshot is created. Record the snapshot number: \_\_\_\_\_
3. Execute the ./workgen 3 1 script to generate the workload for this lab. The script is located in your \$HOME/workshops directory. Wait until the “Load is finished” message appears, and then continue. This script takes about two minutes. Proceed to step 4 immediately after the script finishes.
 

**Note:** The last “\$” prompt does not appear until you press Enter one more time.

### Practice 3-1: Using Automatic Workload Repository (continued)

```
$ ./workgen 3 1

PL/SQL procedure successfully completed.
...
PL/SQL procedure successfully completed.

Load is finished
$
```

4. Take a second AWR snapshot right after workload generation is completed.
  - a) Navigate to the Snapshots page, and then click Create.
  - b) On the Confirmation page, click Yes.
  - c) The processing page appears. Wait until the snapshot is created.
  - d) The Confirmation box appears on the Snapshots page, where you can see that a new snapshot has been created. Record the snapshot number: \_\_\_\_\_
5. Generate an AWR report. You can use the `awrrpt.sql` script located in the `$ORACLE_HOME/rdbms/admin` directory to create an AWR report. The solution in this document uses Enterprise Manager.
  - a) Still on the Snapshots page, select View Report from the Actions drop-down list. Make sure that your first AWR snapshot is selected. This is the snapshot number that you recorded in step 2h. Click Go.
  - b) On the View Report page, make sure that your second AWR snapshot is selected. This is the snapshot that you recorded in step 4d. Click OK.
  - c) The processing page appears. Wait until the report is created.
6. Review the report. What can you determine from this report about the performance of this workload?
  - a) After the report is created, you are taken to the Snapshot Details page where you can see the report.
  - b) Scroll down to Top 5 Timed Events. What wait event accounts for most of the %DB Time? \_\_\_\_\_ (buffer busy waits)

#### Top 5 Timed Foreground Events

Event	Waits	Time(s)	Avg wait (ms)	% DB time	Wait Class
buffer busy waits	1,225	268	219	42.11	Concurrency
DB CPU		59		9.28	
enq: HW - contention	47	54	1153	8.52	Configuration
log file sync	252	34	133	5.27	Commit
free buffer waits	748	21	28	3.25	Configuration

### Practice 3-1: Using Automatic Workload Repository (continued)

- c) Scroll down to Time Model Statistics. Which category collects the most %DB Time? \_\_\_\_\_(sql execute elapsed time)

**Note:** Elapsed time includes wait time.

From these first-level diagnostics, you know that the buffer busy waits occurring during the SQL execution are the cause of the performance issue.

Time Model Statistics		
<ul style="list-style-type: none"><li>• Total time in database user-calls (DB Time): 635.9s</li><li>• Statistics including the word "background" measure background</li><li>• Ordered by % or DB time desc, Statistic name</li></ul>		
Statistic Name	Time (s)	% of DB Time
sql execute elapsed time	564.37	88.75
parse time elapsed	74.92	11.78
hard parse elapsed time	74.07	11.65
DB CPU	59.02	9.28
connection management call elapsed time	18.46	2.90
PL/SQL execution elapsed time	9.49	1.49
PL/SQL compilation elapsed time	5.45	0.86
hard parse (sharing criteria) elapsed time	0.23	0.04
repeated bind elapsed time	0.07	0.01
hard parse (bind mismatch) elapsed time	0.01	0.00
sequence load elapsed time	0.00	0.00
DB time	635.91	
background elapsed time	93.76	
background cpu time	0.76	

- d) The SQL statements that have high elapsed time will be the most likely causes of the waits. You can scroll down to the "SQL ordered by Elapsed Time" section or click the following links under Main Report: SQL Statistics, and then SQL ordered by Elapsed Time. What is the SQL ID of the statements that have the highest percentage of DB time?

---

---

## Practice 3-1: Using Automatic Workload Repository (continued)

### SQL ordered by Elapsed Time

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the code.
- % Total DB Time is the Elapsed Time of the SQL statement divided into the Total Database Time multiplied by 100
- %Total - Elapsed Time as a percentage of Total DB time
- %CPU - CPU Time as a percentage of Elapsed Time
- %IO - User I/O Time as a percentage of Elapsed Time
- Captured SQL account for 82.2% of Total DB Time (s): 906
- Captured PL/SQL account for 69.7% of Total DB Time (s): 906

Elapsed Time (s)	Executions	Elapsed Time per Exec (s)	%Total	%CPU	%IO	SQL Id	SQL Module	SQL Text
603.99	23	26.26	66.70	3.59	0.03	<a href="#">fcu83t10rr413</a>	SQL*Plus	declare t number; begin for t ...
538.15	480,000	0.00	59.43	2.11	0.03	<a href="#">3csh3g3mjhmzh</a>	SQL*Plus	INSERT INTO SPCT VALUES (NULL,...
109.21	28	3.90	12.06	0.02	0.27	<a href="#">4vs91dcv7u1p6</a>	OMS	insert into sys.aud\$( sessioni...
43.29	27	1.60	4.78	0.04	7.88	<a href="#">f711myt0q6cma</a>	perl@edrsr10p1.us.oracle.com (TNS V1-V3)	insert into sys.aud\$( sessioni...
11.32	4	2.83	1.25	1.18	1.28	<a href="#">6gvch1xu9ca3g</a>		DECLARE job BINARY_INTEGER := ...

- e) Click the SQL ID value to see the full SQL text for each of the SQL statements that you identified in step 6d. What do you observe?

**Answer:** The first is a PL/SQL statement with an INSERT statement. The second is the INSERT statement from the same PL/SQL block. The statement inserts records into the SPCT table.

- f) What types of blocks have the buffer busy waits? \_\_\_\_\_  
 Scroll down to the Buffer Wait Statistics report in the Wait Statistics section, or click the following links in succession: Back to Top, Wait Statistics, and Buffer Wait Statistics. (Data blocks; these could be table or index blocks.)

### Buffer Wait Statistics

- ordered by wait time desc, waits desc

Class	Waits	Total Wait Time (s)	Avg Time (ms)
data block	1,764	264	150
segment header	22	3	139
undo header	6	0	80

### Practice 3-1: Using Automatic Workload Repository (continued)

- g) Are the buffer busy waits against one or a few tables and indexes? \_\_\_\_\_  
 If so, which segments are affected? \_\_\_\_\_  
 In what tablespace do these segments exist? \_\_\_\_\_  
 Scroll down to “Segments by Buffer Busy Waits,” or click the following links in succession: Back to Top, Segment Statistics, and Segments by Buffer Busy Waits. (Yes, one table is experiencing most of the waits. The SPCT table is experiencing the waits. The SPCT table is in the TBSSPC tablespace.)

#### Segments by Buffer Busy Waits

- % of Capture shows % of Buffer Busy Waits for each top segment compared
- with total Buffer Busy Waits for all segments captured by the Snapshot

Owner	Tablespace Name	Object Name	Subobject Name	Obj. Type	Buffer Busy Waits	% of Capture
SPC	TBSSPC	SPCT		TABLE	1,779	99.61
SYS	SYSTEM	AUD\$		TABLE	7	0.39

**Diagnostics:** This workload is causing buffer busy waits and block contention by performing inserts from multiple processes into the same table.

**Additional Information:** If the tablespace has segment space management set to manual, all the processes attempt to insert rows into the same small set of blocks. If the segment space management is set to automatic, the processes will insert rows into different blocks, choosing blocks based on a hash of the process\_id.

- h) Check the properties of the TBSSPC tablespace. On the Database home page, click Server, and then Tablespaces. Select the TBSSPC tablespace, and then click View.

#### View Tablespace: TBSSPC

Actions

Name **TBSSPC**  
 Bigfile tablespace **No**  
 Status **ReadWrite**  
 Type **Permanent**  
 Extent Management **local**  
 Encryption **NO**

#### Storage

Allocation Type **Automatic**  
**Segment Space Management **Manual****  
 Enable logging **Yes**  
 Compression **No Compression**  
 Block Size (B) **8192**

**Conclusion:** The TBSSPC tablespace is set to Manual Segment Space Management. Moving the SPC table to a tablespace with Automatic Segment Space Management could make a significant difference in performance.



### Practice 3-1: Using Automatic Workload Repository (continued)

7. To implement the changes to the SPCT table, execute the `wksh_03_01_07.sql` script. This script drops the SPCT table, drops the TBSSPC tablespace, re-creates the tablespace with the ASSM property, and creates the SPCT table. You can locate the `wksh_03_01_07.sql` script in your `$HOME/workshops` directory.

```
$ sqlplus / as sysdba

SQL> @wksh_03_01_07.sql
SQL>
SQL> drop tablespace tbsspc
      2 including contents and datafiles;

Tablespace dropped.

SQL>
SQL> CREATE SMALLFILE TABLESPACE "TBSSPC"
      2 DATAFILE 'tbsspc1.dbf' SIZE 50M
      3 AUTOEXTEND ON NEXT 10M MAXSIZE 200M
      4 LOGGING
      5 EXTENT MANAGEMENT LOCAL
      6 SEGMENT SPACE MANAGEMENT AUTO;

Tablespace created.

SQL>
SQL> connect spc/spc
Connected.
SQL>
SQL> drop table spct purge;
drop table spct purge
      *
ERROR at line 1:
ORA-00942: table or view does not exist

SQL> create table spct(id number, name varchar2(2000))
      2 tablespace tbsspc;

Table created.

SQL>
SQL> exec DBMS_STATS.GATHER_TABLE_STATS(-
> ownname=>'SPC', tabname=>'SPCT',-
> estimate_percent=>DBMS_STATS.AUTO_SAMPLE_SIZE);

PL/SQL procedure successfully completed.

SQL>
SQL> exit;
$
```

### **Practice 3-1: Using Automatic Workload Repository (continued)**

8. Execute the `./workgen 3 1` script again to start generating the workload for this practice. Wait until “Load is finished” appears.

```
$ ./workgen 3 1
...
Load is finished
$
```

9. Take a third AWR snapshot.
  - a) Navigate to the Snapshots page and click Create.
  - b) On the Confirmation page, click Yes.
  - c) The processing page appears. Wait until the snapshot is created.
  - d) The Confirmation box appears on the Snapshots page. You can clearly see that a third snapshot has been created.
10. Generate and review a Compare Periods report comparing the periods between the three snapshots. You can use the `awrddrpt.sql` script located in your `$ORACLE_HOME/rdbms/admin` directory to generate the Compare Periods report. The solution in this document uses Enterprise Manager.
  - a) On the Snapshots page, select Compare Periods from the Actions drop-down list. Make sure that the first AWR snapshot is selected. Click Go.
  - b) On the Compare Periods: First Period End page, select your second AWR snapshot and click Next.
  - c) On the Compare Periods: Second Period Start page, select your second AWR snapshot and click Next.
  - d) On the Compare Periods: Second Period End page, select your last AWR snapshot and click Next.
  - e) On the Compare Periods: Review page, click Finish.
  - f) On the Compare Periods: Results page, select Per Transaction from the View Data drop-down list on the General tabbed page.
  - g) You can see the performance differences between the two analyzed periods. Click Report.
  - h) The processing page appears. Wait until the report is generated.
  - i) On the Report tabbed page, you can now see Workload Repository Compare Period Report.
  - j) In the second section from the top of the report, notice the difference between the Average Active Sessions, and the DB Time between the first and second snapshot sets. Because the workload was identical, the reduction in Average Active Sessions indicates the waits were lower, and the DB Time reduction indicates that less time was spent in database calls. Both indicate that the instance is performing more efficiently.

## Practice 3-1: Using Automatic Workload Repository (continued)

- k) Scroll down to the Top 5 Timed Events section. You can see the difference between the two periods for the buffer busy waits event. (Your results will vary from the following example.) In the second run, the order of the waits will likely change.

### Top Timed Events

- Events with a "-" did not make the Top list in this set of snapshots, but are displayed for comparison purposes

1st						2nd					
Event	Wait Class	Waits	Time(s)	Avg Time(ms)	%DB time	Event	Wait Class	Waits	Time(s)	Avg Time(ms)	%DB time
buffer busy waits	Concurrency	1,225	267.76	218.58	42.11	buffer busy waits	Concurrency	180	85.51	475.07	16.89
CPU time			59.02		9.28	log file switch (private strand flush incomplete)	Configuration	22	74.17	3,371.43	14.65
enq: HW - contention	Configuration	47	54.21	1,153.43	8.52	free buffer waits	Configuration	90	68.18	757.53	13.47
db file async I/O submit	System I/O	29	35.02	1,207.66	5.51	log file sync	Commit	169	62.81	371.67	12.41
log file sync	Commit	252	33.52	133.00	5.27	CPU time			49.14		9.71
-free buffer waits	Configuration	748	20.65	27.61	3.25	-db file async I/O submit	System I/O	148	25.94	175.28	5.12
-log file switch (private strand flush incomplete)	Configuration	7	12.71	1,816.05	2.00	-					

- l) Scroll down to the Buffer Wait Statistics in the Wait Stats section. Here also, you can clearly see the difference.

### Buffer Wait Statistics

- Ordered by absolute value of 'Diff' column of 'Wait Time % of DB time' descending

Class	Wait Time % of DB time			Total Wait Time (s)		# Waits		Avg Wait Time (ms)		
	1st	2nd	Diff	1st	2nd	1st	2nd	1st	2nd	%Diff
data block	0.42	0.16	-0.25	264.37	82.65	1,764	300	1.50	2.76	84.00
undo header	0.00	0.00	0.00	0.48	2.09	6	5	0.80	4.18	422.50
segment header	0.00	0.00	-0.00	3.05	0.74	22	6	1.39	1.23	-11.51

### Practice 3-1: Using Automatic Workload Repository (continued)

- m) Scroll down to the Top Segments by Buffer Busy Waits in the Segment Statistics section. The number of buffer busy waits has changed. Possibly, the classes of buffer busy waits have changed as well. (This can be seen in some reports where one period or the other is reporting 0 waits.) The bitmap block (BMB) class of wait cannot occur in the first period because the BMB block exists only in the ASSM tablespaces.

#### Top Segments by Buffer Busy Waits

- Ordered by absolute value of 'Diff' column of '% of Total Buffer Busy Waits'
- Top 5 Segments from each period are compared
- Total Buffer Busy Waits First: 1,225, Second: 180
- Buffer Busy Wait Time First: 267.76 seconds, Second: 85.51 seconds
- Buffer Busy Wait Time as % of DB time First: 42.11%, Second: 16.89%
- Captured Buffer Busy Waits First: 1,786, Second: 306

Owner	Tablespace	Object Name	Subobject Name	Type	% of Total Buffer Busy Waits					Buffer Busy Waits			% of Captured Buffer Busy Waits		
					1st	1st Total	2nd	2nd Total	Diff	1st	2nd	%Diff	1st	2nd	Diff
SPC	TBSSPC	SPCT		TABLE	145.22	145.22	0.00	0.00	-145.22	1,779	0	-100.00	99.61	0.00	-99.61
SYS	SYSTEM	AUD\$		TABLE	0.57	145.80	24.44	24.44	23.87	7	44	528.57	0.39	14.38	13.99
SPC	TBSSPC	SPCT		TABLE	145.22	291.02	145.56	170.00	0.33	1,779	262	-85.27	99.61	85.62	-13.99

11. Run the `./cleanup 3 1` cleanup script.

```
$ ./cleanup 3 1
```

## Practices for Lesson 4

The goal of this practice is to identify the sources of performance issues.

## Practice 4-1: Using Enterprise Manager to Identify OS Issues

In this practice, the workload generator starts an application load on the database and an OS load.

1. In a terminal window, use the `date` command to determine the start time of the workload.

```
$ date
Tue Mar 25 06:19:57 GMT-7 2008
```

2. In a terminal window, change directory to `/home/oracle/workshops`. Start the workload generator with the command: `./workgen 4 1`. Verify that the workload generator has started the workload processes by executing the `ps` command.

```
$ cd /home/oracle/workshops
$ ./workgen 4 1
$ ps
  PID TTY          TIME CMD
 2365 pts/2        00:00:00 bash
 15312 pts/2        00:00:00 wkld_shell.sh
 15313 pts/2        00:00:00 insert_orders.s
 15314 pts/2        00:00:00 update_orders.s
 15317 pts/2        00:00:00 delete_orders.s
 15318 pts/2        00:00:00 sqlplus
 15319 pts/2        00:00:00 insert_orders.s
 15320 pts/2        00:00:00 sqlplus
 15321 pts/2        00:00:00 update_orders.s
 15322 pts/2        00:00:00 delete_orders.s
 15323 pts/2        00:00:00 sqlplus
 15329 pts/2        00:00:00 insert_orders.s
 15330 pts/2        00:00:00 sqlplus
 15445 pts/2        00:00:00 sqlplus
 15452 pts/2        00:00:00 sqlplus
 15459 pts/2        00:00:00 sleep
 15460 pts/2        00:00:00 sleep
 15461 pts/2        00:00:00 sleep
 15462 pts/2        00:00:00 sleep
 15463 pts/2        00:00:00 sleep
 15464 pts/2        00:00:00 sleep
...
```

3. Use Enterprise Manager to determine the source of the loads. In the browser's address bar, enter the URL, <https://<hostname>:1158/em>, to view the `orcl` database information. On the Login page, enter `SYS` as the username and `oracle` as the password, select `SYSDBA` from the Connect As drop-down list, and click Login.
4. Wait for a while until the Active Sessions graph shows some waits. Find the OS process that is consuming the most CPU.

## Practice 4-1: Using Enterprise Manager to Identify OS Issues (continued)

- The Database home page for the `orcl.us.oracle.com` database appears. Observe the Host CPU measurement.
  - A significant percentage of the Host CPU is dedicated to **Other** processes. Click the Other link to view the Performance Summary page.
  - Scroll down to view Top 10 Processes by specifying View By CPU Utilization (%).
- Note:** Refresh this view a few times until an OS process (not owned by `oracle`) appears. The `dd` process should appear. It may not be the top process. What is the process ID of the `dd` process?

**Processes**  
Processes [244](#)

**Top 10 Processes**  
View By: CPU Utilization (%)

Process ID	Command	CPU Utilization (%)	CPU Total (seconds)	Resident Size (KB)	Virtual Size (KB)	Owner
16321	oracleorcl (DESCRIPTION=(LOCAL=YES)(ADDRESS=(PROTOCOL=beq)))	1.75	0	21,396	949,720	oracle
16324	oracleorcl (DESCRIPTION=(LOCAL=YES)(ADDRESS=(PROTOCOL=beq)))	1.35	0	21,996	949,720	oracle
14705	dd if /dev/sda5 of /dev/null conv ebodic bs 102400 count 23520	0.85	5	624	4,188	root
16311	oracleorcl (DESCRIPTION=(LOCAL=YES)(ADDRESS=(PROTOCOL=beq)))	0.75	0	22,652	949,720	oracle

- Stop the `dd` process using the `kill` command. Because this process is running as `root`, use the following command:

```
$ sudo kill <process ID>
```

- Stop the workload by executing the `rm runload` command.

```
$ rm runload
```

In this practice, you found and stopped an OS process that was consuming CPU.

## Practices for Lesson 5

The goal of this practice is to create and monitor a server-generated alert.



## Practice 5-1: Using Metrics

This practice shows you how to set up server-generated alerts and how to retrieve alert information from Enterprise Manager. During this practice, you see how alerts are triggered. However, this practice does not involve analysis of the cause of the generated alerts. This analysis is performed during subsequent practices by using AWR snapshots and ADDM.

1. Change to the `/home/oracle/workshops` directory. Execute the `./prepare 5 1` script to set up the necessary objects before starting the workload. Log out of Enterprise Manager.

```
$ cd /home/oracle/workshops
$ ./prepare 5 1
Database closed.
Database dismounted.
ORACLE instance shut down.
ORACLE instance started.

Total System Global Area  836976640 bytes
Fixed Size                  1339740 bytes
Variable Size              805310116 bytes
Database Buffers           25165824 bytes
Redo Buffers                5160960 bytes
Database mounted.
Database opened.
drop tablespace tbsjfv including contents and datafiles
*
ERROR at line 1:
ORA-00959: tablespace 'TBSJFV' does not exist

Tablespace created.

Table created.

PL/SQL procedure successfully completed.

User dropped.

User created.

Grant succeeded.

drop table jfvvt purge
*
```

## Practice 5-1: Using Metrics (continued)

```
ERROR at line 1:
ORA-00942: table or view does not exist

Table created.

PL/SQL procedure successfully completed.

PL/SQL procedure successfully completed.

$
```

2. Using EM Database Control for the `orcl` instance, determine the current threshold values set for the `DB_TIME_WAITING` metric. Pay special attention to the Concurrency wait class. Use both Enterprise Manager and SQL\*Plus to find those values.
  - a) On the Database home page, scroll down to the Related Links section, and click All Metrics.

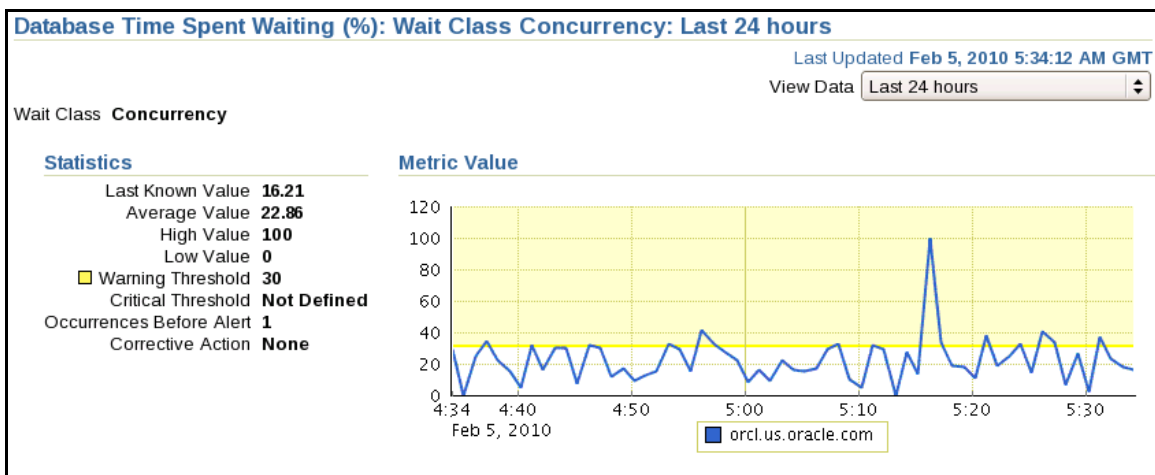
Related Links		
<a href="#">Access</a>	<a href="#">Add Exadata Cell Targets</a>	<a href="#">Advisor Central</a>
<a href="#">Alert History</a>	<a href="#">Alert Log Contents</a>	<a href="#">All Metrics</a>
<a href="#">Baseline Metric Thresholds</a>	<a href="#">Blackouts</a>	<a href="#">EM SQL History</a>
<a href="#">Jobs</a>	<a href="#">Metric and Policy Settings</a>	<a href="#">Metric Collection Errors</a>
<a href="#">Monitoring Configuration</a>	<a href="#">Monitor in Memory Access Mode</a>	<a href="#">Policy Groups</a>
<a href="#">Scheduler Central</a>	<a href="#">SQL Worksheet</a>	<a href="#">Target Properties</a>
<a href="#">User-Defined Metrics</a>		

- b) On the All Metrics page, scroll to Waits by Wait Class and expand it.
  - c) Click Database Time Spent Waiting (%).
  - d) On the Database Time Spent Waiting (%) page, you can see all thresholds set for each class. Click Concurrency.

## Practice 5-1: Using Metrics (continued)

Database Time Spent Waiting (%)					
				Latest Data Collected From Target <b>Feb 5, 2010 5:16:52 AM GMT</b>	
				View Data	Last 24 hours
Wait Class	Average Value	High Value	Low Value	Last Known Value	Collection Timestamp
Cluster	0	0	0	0	Feb 5, 2010 5:30:12 AM
Application	1.41	43.92	0.02	0.02	Feb 5, 2010 5:30:12 AM
Idle	0	0	0	0	Feb 5, 2010 5:30:12 AM
System I/O	15.87	100	1.27	4.18	Feb 5, 2010 5:30:12 AM
User I/O	2.02	33.23	0	33.23	Feb 5, 2010 5:30:12 AM
Network	0.14	0.81	0.03	0.1	Feb 5, 2010 5:30:12 AM
Scheduler	0	0	0	0	Feb 5, 2010 5:30:12 AM
Queueing	0	0	0	0	Feb 5, 2010 5:30:12 AM
Configuration	0	0	0	0	Feb 5, 2010 5:30:12 AM
Commit	7.25	39.47	0.12	3.75	Feb 5, 2010 5:30:12 AM
Administrative	0	0	0	0	Feb 5, 2010 5:30:12 AM
Other	0.86	11.9	0.03	0.03	Feb 5, 2010 5:30:12 AM
Concurrency	22.44	100	0	2.22	Feb 5, 2010 5:30:12 AM

- e) View the Wait Class Concurrency page.  
 What is the default Warning threshold? \_\_\_\_\_  
 Have there been any alerts generated? \_\_\_\_\_  
 Do the alerts correspond to the graph? \_\_\_\_\_ Check the alert history below the graph. The graph does not display information before the last database restart.



- f) From a SQL\*Plus session, determine the threshold values for the wait classes shown in the preceding screenshot. Change the directory to /home/oracle/labs. Connect as a sysdba user and execute the lab\_05\_01\_02.sql script.

```
$ cd /home/oracle/labs
$ sqlplus / as sysdba

SQL> @lab_05_01_02.sql
SQL>
SQL> variable wo number
SQL> variable wv varchar2(10)
```

**Practice 5-1: Using Metrics (continued)**

```

SQL> variable co number
SQL> variable cv varchar2(10)
SQL> variable op number
SQL> variable ct number
SQL>
SQL> BEGIN
  2  DBMS_SERVER_ALERT.get_threshold(
  3      metrics_id => dbms_server_alert.DB_TIME_WAITING,
  4      warning_operator      => :wo,
  5      warning_value         => :wv,
  6      critical_operator     => :co,
  7      critical_value        => :cv,
  8      observation_period    => :op,
  9      consecutive_occurrences => :ct,
 10      instance_name         => 'orcl',
 11      object_type =>
 12          dbms_server_alert.OBJECT_TYPE_EVENT_CLASS,
 13      object_name           => 'Concurrency');
 14  END;
 15  /

```

PL/SQL procedure successfully completed.

```

SQL>
SQL> print  wv

```

```

WV
-----
30

```

```

SQL>
SQL> col object_name format a20
SQL> col metrics_name format a25
SQL> col warning_value format a10
SQL> col critical_value format a10
SQL>
SQL> select object_name,warning_value,critical_value
  2  from dba_thresholds
  3  where metrics_name='Database Time Spent Waiting (%)';

```

OBJECT_NAME	WARNING_VA	CRITICAL_V
Concurrency	30	
Administrative	30	
Application	30	
Configuration	30	
Network	30	
Other	30	
Cluster	50	
Commit	50	

8 rows selected.

## Practice 5-1: Using Metrics (continued)

```
SQL>
SQL> exit;
$
```

3. Using a PL/SQL command or the lab\_05\_01\_03.sql script, change the thresholds for the Concurrency class to 60% for the warning level and 90% for the critical level. Using both SQL\*Plus and Enterprise Manager, verify that your change took effect.
  - a) Use the DBMS\_SERVER\_ALERT package to set the new threshold values.

```
$ sqlplus / as sysdba
```

```
SQL> @lab_05_01_03.sql
```

```
SQL>
```

```
SQL> BEGIN
```

```
 2     DBMS_SERVER_ALERT.set_threshold(
 3     metrics_id => dbms_server_alert.DB_TIME_WAITING,
 4     warning_operator => dbms_server_alert.OPERATOR_GE,
 5     warning_value    => 60,
 6     critical_operator=> dbms_server_alert.OPERATOR_GE,
 7     critical_value   => 90,
 8     observation_period=> 1,
 9     consecutive_occurrences => 1,
10     instance_name    => 'orcl',
11     object_type       =>
12         dbms_server_alert.OBJECT_TYPE_EVENT_CLASS,
13     object_name       => 'Concurrency');
14     END;
15 /
```

PL/SQL procedure successfully completed.

```
SQL>SQL> col object_name format a20
```

```
col metrics_name format a25
```

```
col warning_value format a10
```

```
col critical_value format a10
```

```
SQL> SQL> SQL> SQL>
```

```
SQL> select object_name,warning_value,critical_value
 2     from dba_thresholds
 3     where metrics_name=
 4         'Database Time Spent Waiting (%)';
```

OBJECT_NAME	WARNING_VA	CRITICAL_V
Concurrency	60	90
Administrative	30	
Application	30	
Configuration	30	
Network	30	
Other	30	

## Practice 5-1: Using Metrics (continued)

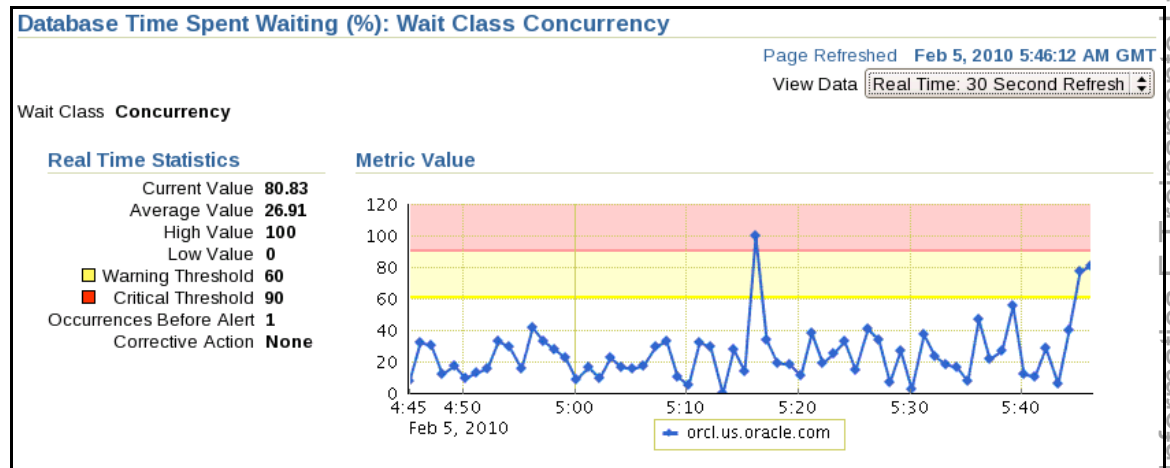
```
Cluster          50
Commit           50
```

8 rows selected.

```
SQL> exit;
```

```
$
```

- b) Still on the Wait Class Concurrency page, select Real Time: 30 Second Refresh from the View Data drop-down list. Note that the Warning and Critical values have been changed.



4. Change the directory to \$HOME/workshops. Execute the ./workgen 5 1 script to start the workload for this practice. While the script is executing, observe the variation of the curve on the Wait Class Concurrency page. After the workload is finished, find the alert history from a SQL\*Plus session. Check both the outstanding alerts and the alert history. The workload script takes up to six minutes to complete on some machines.

```
$ cd $HOME/workshops
```

```
$ ./workgen 5 1
```

```
$
```

... The following messages will appear for 1 to 6 minutes ...

```
PL/SQL procedure successfully completed.
```

```
PL/SQL procedure successfully completed.
```

```
PL/SQL procedure successfully completed.
```

```
PL/SQL procedure successfully completed.
```

```
PL/SQL procedure successfully completed.
```

### ***Practice 5-1: Using Metrics (continued)***

PL/SQL procedure successfully completed.

PL/SQL procedure successfully completed.

PL/SQL procedure successfully completed.

PL/SQL procedure successfully completed.

PL/SQL procedure successfully completed.

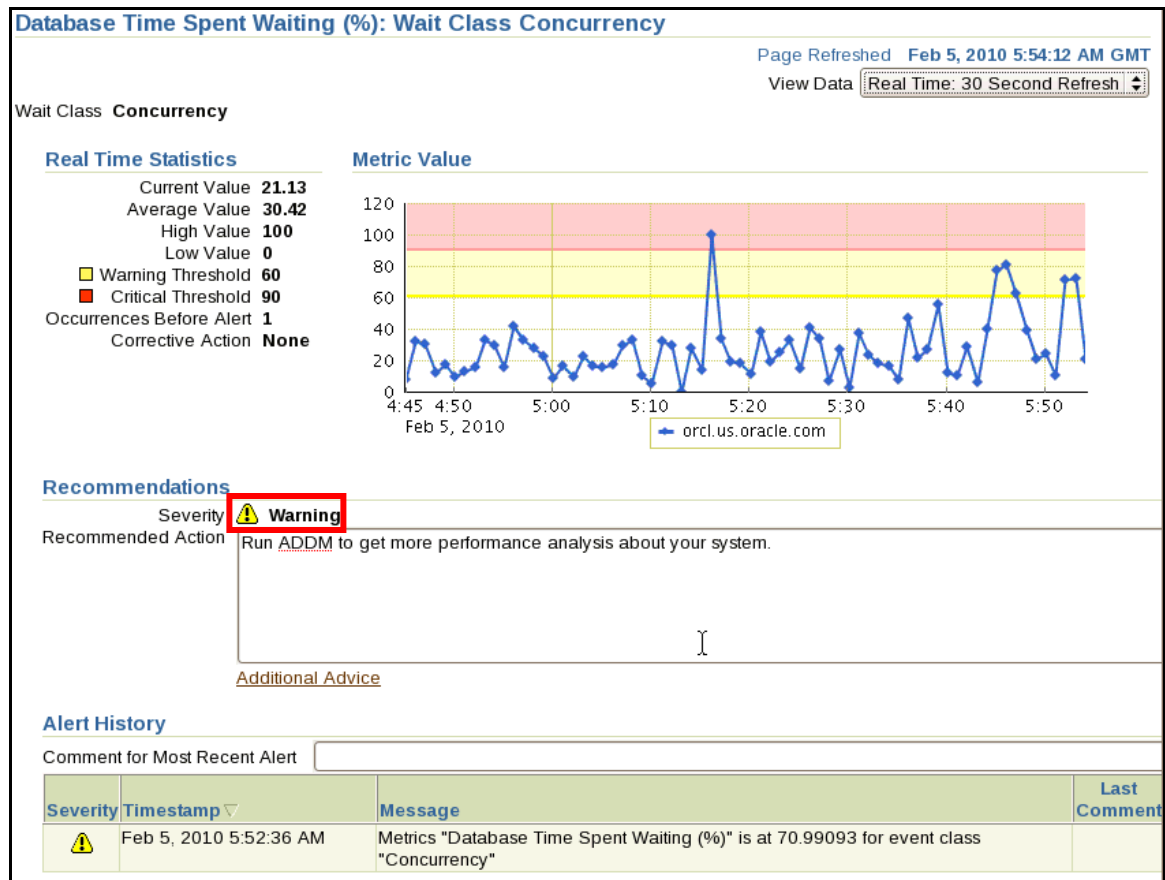
PL/SQL procedure successfully completed.

PL/SQL procedure successfully completed.

\$

## Practice 5-1: Using Metrics (continued)

- While the script executes, look at the evolution of the curve on the Wait Class Concurrency graph. After a while, you should see a warning or critical alert.



- After the script finishes, you can retrieve the history of your alerts by using SQL\*Plus. Depending on your timing, the alert may appear in the DBA\_OUTSTANDING\_ALERTS or DBA\_ALERT\_HISTORY view. If the alert is not yet cleared, it shows in outstanding Alerts view. After it is cleared, it is visible in the Alert History view. The following example shows the output when the alert has been cleared (**Hint:** lab\_05\_01\_06.sql in the \$HOME/labs directory).

```
$ sqlplus / as sysdba

SQL> @../labs/lab_05_01_06.sql
SQL>
SQL> col reason format a75
SQL>
SQL> SELECT reason FROM DBA_OUTSTANDING_ALERTS;

REASON
-----
Metrics "Database Time Spent Waiting (%)" is at 63.59094 for
event class "Concurrency"
```



**Practice 5-1: Using Metrics (continued)**

```

SQL>
SQL> SELECT TO_CHAR(begin_time,'hh24:mi:ss'),
2         dbtime_in_wait,average_waiter_count
3 FROM V$WAITCLASSMETRIC_HISTORY
4 WHERE wait_class#=4
5 AND wait_class_id=3875070507
6 AND begin_time > SYSDATE-(10/1440);

TO_CHAR( DBTIME_IN_WAIT AVERAGE_WAITER_COUNT
-----
05:55:12      63.5909417          .001635947
05:54:12      24.6778322          .004029849
05:53:11      21.1284229          .055115698
05:52:11      71.9247151         13.4691388
05:51:12       70.990928          7.82854524
05:50:12      10.3871741          .001729537
05:49:12      24.3581563          .003170473
05:48:11      20.7383737          .001333433
05:47:11      38.9326013          .001199334

9 rows selected.

SQL>
SQL>
SQL> -- Cleared!
SQL>
SQL> SELECT reason,resolution
2 FROM DBA_ALERT_HISTORY
3 WHERE reason like
4      '%Database Time Spent Waiting (%)%Concurrency%'
5 AND TO_DATE(SUBSTR(TO_CHAR(creation_time),1,18)||
6      SUBSTR(TO_CHAR(creation_time),26,3) ,
7      'DD-MON-YY HH:MI:SS AM') > SYSDATE-(10/1440)
8 ORDER BY creation_time DESC;

REASON
-----
RESOLUT
-----
Metrics "Database Time Spent Waiting (%)" is at 21.12842 for
event class "Concurrency"
cleared

SQL>
SQL> EXIT;

$

```

## Practice 5-1: Using Metrics (continued)

7. Check Alert History from EM.
  - a) In the Related Links section of the Database home page, click Alert History. Are the alerts that you generated in this practice visible?
  - b) Click the Database Time Spent Waiting (%) link.  
**Note:** This action displays Database Time Spent Waiting (%) with the wait by class.
  - c) Click the Concurrency link.  
**Note:** This action displays the Database Time Spent Waiting (%): Wait Class Concurrency page with the graph showing the waits and thresholds.
8. In a terminal window, change directory to `/home/oracle/workshops` and execute the `cleanup 5 1` script to clean up your environment.

```
$ ./cleanup 5 1

User dropped.

Tablespace dropped.

PL/SQL procedure successfully completed.

$
```

## Practices for Lesson 6

The goal of this practice is to create a static baseline and a repeating baseline. The static baseline is used to create an adaptive threshold on a metric.

## Practice 6-1: Using Baselines

In this practice, you create a static baseline over the snapshots that are present in the database, from 9:00 PM the day before this class started to 5:00 AM the following morning. This set of snapshots should be in the range of 1–100. (Your instructor may have different instructions). Name the baseline LAST\_NIGHT.

1. Create the LAST\_NIGHT baseline.

Step	Page	Action		
a	Database Instance home	Click the Server tab.		
b	Server	Click Automatic Workload Repository in the Statistics Management section.		
c	Automatic Workload Repository	Click the Snapshots link.		
d	Snapshots	Write the time and date of snapshots 2 and 100 (or the most recent, whichever is the least).	Date	Time
		2		
		100 (most recent)		
		Click Automatic Workload Repository in the navigation path at the top of the page to return to the Automatic Workload Repository page.		
e	Automatic Workload Repository	Click the Baselines link.		
f	AWR Baselines	Click Create.		
g	Create Baseline:	Select Single.		

**Practice 6-1: Using Baselines (continued)**

	Baseline Interval Type	Click Continue.		
h	Create Baseline: Single Baseline		<b>Field</b>	<b>Value</b>
		Complete the baseline name field.	Baseline Name	LAST_NIGHT
		Select Snapshot Range. Move the chart view range until you can see the set of snapshots 2 through 100.		
		Select the start and end snapshots.	Period Start Time	Time of Snapshot 2 Or as directed by the instructor
			Period End Time	Time of snapshot 100 Or as directed by the instructor
		Click Finish.		

2. On the AWR Snapshots page, calculate the statistics for the LAST\_NIGHT baseline.

Step	Page	Action
a.	AWR Baselines	Select LAST_NIGHT.
		From the Actions drop-down list, select Schedule Statistics Computation.
		Click Go.
b	Compute Threshold Statistics: LAST_NIGHT	Select Immediately.
		Click Submit.

3. Set the Performance page to show the LAST\_NIGHT baseline statistics.
- a) Click Settings on the Performance page.

### Practice 6-1: Using Baselines (continued)

- b) On the Performance Page Settings page, in the Baseline display section, set the baseline to LAST\_NIGHT.

Select “Show the 99th percentile line using a static baseline with computed statistics.” Click OK.

Step	Page	Action
a	AWR Baselines	Click the Database tab.
b	Database Home	Click the Performance tab.
c	Performance	Click Settings.
d	Performance Page Settings	From the Baseline Name drop-down list, select LAST_NIGHT.
		Select “Show the 99th percentile line using a static baseline with computed statistics.”
		Click OK.

4. Create a template that creates a baseline for the Monday maintenance window for the next month. Keep the baseline for 60 days. The weekday maintenance window start and end times can be found on the Windows page Home > Server > Windows (in the Scheduler section).

Step	Page	Action
a.	Performance	Click the Server tab.
b	Server	Click Windows in the Oracle Scheduler section.
c	Scheduler Windows	Record the start time of MONDAY_WINDOW. This is shown as Repeat Time (should be 10:00 PM).
		Record the duration of MONDAY_WINDOW.
		Click the Database tab.
d	Database home	Click the Server tab.
e	Server	Click AWR Baselines.

## Practice 6-1: Using Baselines (continued)

f	AWR Baselines	Click Create.		
g	Create Baseline: Baseline Interval Type	Click Repeating.		
		Click Continue.		
h	Create Baseline: Repeating Baseline Template (See the example in the following screenshot.)	Set values on the page:	<b>Field</b>	<b>Value</b>
			Baseline Name Prefix	MONDAY
			Baseline Time Period: Start Time	10 PM
			Baseline Time Period: Duration	4
			Frequency	Weekly and Monday
			Interval of Baseline Creation: Start Time	Today 9:50 PM
			Interval of Baseline Creation: End Time	One Month from today 2:20 AM
			Retention Time (days)	60
		Click Finish.		
i	AWR Baselines	Click AWR Baseline Templates in the Related Links section.		
j	AWR Baseline Templates	Verify that the MONDAY template was created as expected.		

# Practice 6-1: Using Baselines (continued)

h)

Create Baseline: Repeating Baseline Template

Cancel
Back
Finish

The repeating type of baseline has a time interval that repeats over a time period. For example, every Monday from 10:00 AM to 12:00 PM for the year 2007.

\* Baseline Name Prefix

Baseline Time Period

Start Time   Duration (Hours)

Frequency

☐ Daily  
☒ Weekly  
☒ Monday ☐ Tuesday ☐ Wednesday ☐ Thursday ☐ Friday ☐ Saturday ☐ Sunday

Interval of Baseline Creation

Start Time     End Time

Purge Policy

Retention Time (Days)

**TIP** A baseline template with the same name as the baseline name prefix will be created.

- Define an alert threshold for transactions per second based on the LAST\_NIGHT baseline. Set the critical threshold to 120% of maximum and the warning to 110% of maximum.

Step	Page	Action		
a	AWR Baseline Templates	Click AWR Baselines in the Related Links section.		
b	AWR Baselines	Click Baseline Metric Thresholds in the Related Links section.		
c	Baseline Metric Thresholds	Change the view to Basic Metrics.		
		Edit the Number of Transaction (per second) metric. (Click the pencil icon on the right.)		
d	Edit Thresholds: Number of Transactions (per second)	Set values:	Field	Value
			Name	LAST_NIGHT
			Threshold Type	Set to Percentage of Maximum
			Critical	120



**Practice 6-1: Using Baselines (continued)**

			Warning	110
			Occurrences	1
		Click Preview. To move the active window slider (in the upper left) over the period of time that covers the LAST_NIGHT baseline, click the day that has the majority of the baseline statistics in it.		
		Write down the estimated number of transactions per second required to produce a critical alert:_____		
		Click Apply Thresholds.		

6. Run a workload by executing the `./workgen 6 1` script. The script will ask for the number of transactions per second that is required to produce a critical alert. If the script is running correctly, you can run the `ps` command and see the `baseline_wkld.sh` process.

```
$ cd /home/oracle/workshops
$ ./workgen 6 1
Enter the number of transactions needed to produce a critical
alert:12
$ ps
  PID TTY          TIME CMD
 29422 pts/2    00:00:00 bash
 31484 pts/2    00:00:00 baseline_wkld.sh
 31511 pts/2    00:00:00 sleep
 31512 pts/2    00:00:00 ps
```

7. On Database Home page, set View Data to Automatically (15 sec). After a short time, an alert appears in the Alerts section of the Database home page of EM. This alert is named “Number of Transactions (per second)” and the message indicates the value of the metric.

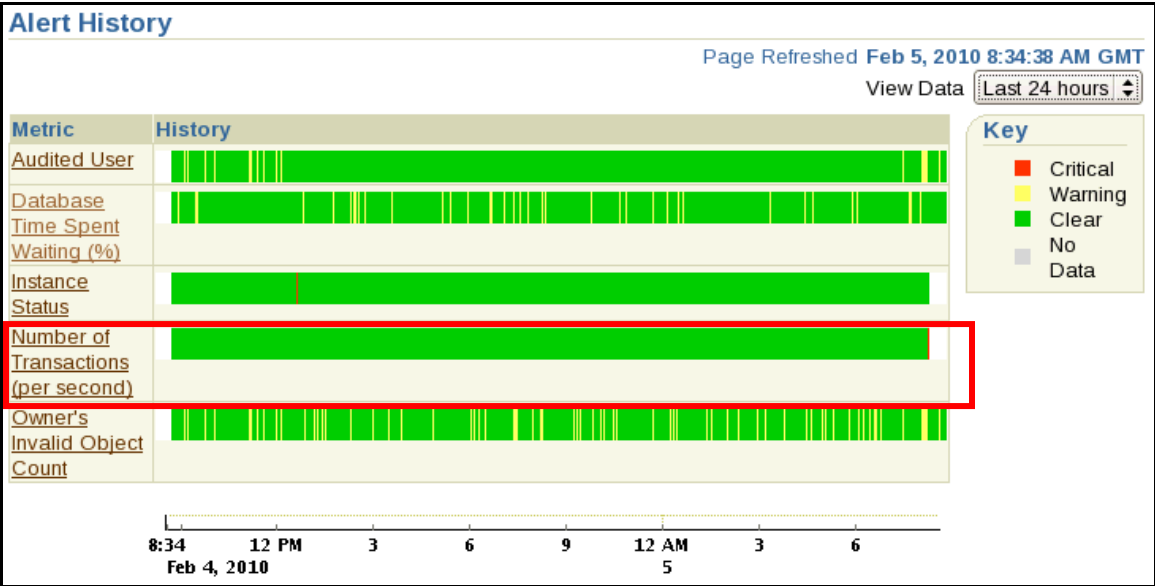
Alerts					
Category		All	Go	Critical <span style="color:red">✖</span> 1	Warning <span style="color:yellow">⚠</span> 2
Severity	Category	Name	Impact	Message	Alert Triggered
<span style="color:red">✖</span>	Throughput	Number of Transactions (per second)		Metrics "User Transaction Per Sec" is at 34.80939	Feb 5, 2010 8:31:44 AM
<span style="color:yellow">⚠</span>	User Audit	Audited User		User SYS logged on from edrsr10p1.us.oracle.com.	Feb 5, 2010 7:21:24 AM
<span style="color:yellow">⚠</span>	Invalid Objects by Schema	Owner's Invalid Object Count		6 object(s) are invalid in the OE schema.	Jan 27, 2010 11:04:11 AM

Stop the workload by deleting the runload file from the `/home/oracle/workshops` directory.

**Practice 6-1: Using Baselines (continued)**

```
$ cd /home/oracle/workshops
$ rm runload
```

8. A short while later, the alert is cleared and no longer appears on the home page. The alert history shows that this metric had one or more critical alerts. View the alert history for this metric.
- a) Click Alert History in the Related Links section of the home page.



9. Clean up this practice by removing the adaptive threshold.

Step	Page	Action
a	Database Home page	Click Baseline Metric Thresholds in the Related Links section.
b	Baseline Metric Thresholds	Click “Number of Transactions (per second).”
c	Edit Thresholds: Number of Transactions (per second) page	Click Clear Thresholds.



## Practice 7-1: Using AWR-Based Tools

In this practice, you use AWR-based tools to find and tune a database problem. Note that during this practice, you explicitly capture AWR snapshots. However, by default, the system automatically captures AWR snapshots every hour.

1. Execute the `./prepare 7 1` script to set up this practice. You can find this script in your `$HOME/workshops` directory. The script is displayed here to show the type of segment management that is specified for the tablespace.

```
$ cd $HOME/workshops
$ ./prepare 7 1

SQL> Connected.
SQL> Database closed.
Database dismounted.
ORACLE instance shut down.
ORACLE instance started.

Total System Global Area  836976640 bytes
Fixed Size                  1339740 bytes
Variable Size              805310116 bytes
Database Buffers           25165824 bytes
Redo Buffers                5160960 bytes
Database mounted.
Database opened.
SQL> drop tablespace tbsspc including contents and datafiles;

Tablespace dropped.

SQL>
SQL> CREATE SMALLFILE TABLESPACE "TBSSPC"
  2  DATAFILE 'tbsspc1.dbf' SIZE 50M
  3  AUTOEXTEND ON NEXT 10M MAXSIZE 200M
  4  LOGGING
  5  EXTENT MANAGEMENT LOCAL
  6  SEGMENT SPACE MANAGEMENT MANUAL;

Tablespace created.

SQL>
SQL> execute
dbms_workload_repository.modify_snapshot_settings(interval =>
1440);

PL/SQL procedure successfully completed.

SQL>
SQL> exec
dbms_advisor.set_default_task_parameter('ADDM','DB_ACTIVITY_MI
N',30);

PL/SQL procedure successfully completed.
```

**Practice 7-1: Using AWR-Based Tools (continued)**

```

SQL>
SQL> drop user spc cascade;

User dropped.

SQL>
SQL> create user spc identified by spc
      2 default tablespace tbsspc
      3 temporary tablespace temp;

User created.

SQL>
SQL> grant connect, resource, dba to spc;

Grant succeeded.

SQL>
SQL> connect spc/spc
Connected.
SQL>
SQL> drop table spct purge;
drop table spct purge
      *
ERROR at line 1:
ORA-00942: table or view does not exist

SQL> create table spct(id number, name varchar2(2000));

Table created.

SQL>
SQL> exec DBMS_STATS.GATHER_TABLE_STATS(-
> ownname=>'SPC', tabname=>'SPCT',-
> estimate_percent=>DBMS_STATS.AUTO_SAMPLE_SIZE);

PL/SQL procedure successfully completed.

SQL>
SQL> exit;
$

```

2. Take the first AWR snapshot. You can use either Enterprise Manager or the DBMS\_WORKLOAD\_REPOSITORY package.

Step	Page	Action
a	EM Database home	Click the Server tab.

**Practice 7-1: Using AWR-Based Tools (continued)**

b	Server	Click Automatic Workload Repository.
c	Automatic Workload Repository	Click the Snapshots link.
d	Snapshots	Click Create.
e	Confirmation	Click Yes.
f	Processing	Wait until the snapshot is created.
g	Snapshots	Record the new snapshot number: _____

- Execute the `./workgen 7 1` script to generate the workload for this practice. The script is located in your `$HOME/workshops` directory. Wait until the “Load is finished” message appears, and then continue.

```
$ ./workgen 7 1
```

```
PL/SQL procedure successfully completed
```

```
...
```

```
Load is finished
```

```
$
```

- Take a second AWR snapshot. You can use the `sol_07_01_02.sh` script located in your `$HOME/solutions` directory to take an AWR snapshot by using the `DBMS_WORKLOAD_REPOSITORY` package, or you can use Enterprise Manager to take a snapshot. Repeat the steps in step 2. Record the snapshot number: \_\_\_\_\_
- Generate the AWR report. You can use the `awrrpt.sql` script located in the `$ORACLE_HOME/rdbms/admin` directory to create an AWR report. The solution in this document uses Enterprise Manager.

Step	Page	Action
a	Snapshots	Select the first snapshot number from step 2.
		Select View Report from Actions.
		Click Go.
b	View Report	Select the second snapshot number from step 4.
		Click OK.

## Practice 7-1: Using AWR-Based Tools (continued)

c	Processing:View Report	Wait until the report is created.
d	Snapshot Details	

6. Review the AWR Report.

- a) What is the DB time? \_\_\_\_\_  
 What is the elapsed time? \_\_\_\_\_  
 Why is there a difference? \_\_\_\_\_

(DB Time includes the sum of all sessions in database calls, so DB Time could be as high as the product of the active sessions and the elapsed time.)

Snapshot Details

View A

Details

Report

58

WORKLOAD REPOSITORY report for

DB Name	DB Id	Instance	Inst num	Startup Time	Release	RAC
ORCL	1237161768	orcl	1	05-Feb-10 09:02	11.2.0.1.0	NO

Host Name	Platform	CPUs	Cores	Sockets	Memory (GB)
edrsr10p1.us.oracle.com	Linux IA (32-bit)	1	1	1	1.97

	Snap Id	Snap Time	Sessions	Cursors/Session
Begin Snap:	277	05-Feb-10 09:05:45	34	2.1
End Snap:	278	05-Feb-10 09:10:17	37	2.8
Elapsed:		4.54 (mins)		
DB Time:		13.46 (mins)		

- b) Scroll down to Top 5 Timed Foreground Events. (Or, use the Find command in your browser.)  
 What is the top event? \_\_\_\_\_  
 ("buffer busy waits" is expected.)

**Practice 7-1: Using AWR-Based Tools (continued)****Top 5 Timed Foreground Events**

Event	Waits	Time(s)	Avg wait (ms)	% DB time	Wait Class
buffer busy waits	877	398	454	49.24	Concurrency
log file sync	170	83	489	10.29	Commit
db file sequential read	4,835	52	11	6.43	User I/O
DB CPU		44		5.44	
free buffer waits	1,961	35	18	4.38	Configuration

- c) Scroll down to Time Model Statistics. (Or, select Main Report > Wait Event Statistics > Time Model Statistics.)  
In which category is most of the DB time being collected?

(“sql execute elapsed time” is expected.)

**Time Model Statistics**

- Total time in database user-calls (DB Time): 807.9s
- Statistics including the word "background" measure background time statistic
- Ordered by % or DB time desc, Statistic name

Statistic Name	Time (s)	% of DB Time
sql execute elapsed time	711.88	88.12
parse time elapsed	66.64	8.25
hard parse elapsed time	66.04	8.17
DB CPU	43.95	5.44
connection management call elapsed time	31.81	3.94
PL/SQL compilation elapsed time	30.81	3.81
PL/SQL execution elapsed time	12.72	1.57
hard parse (sharing criteria) elapsed time	0.15	0.02
repeated bind elapsed time	0.06	0.01
hard parse (bind mismatch) elapsed time	0.00	0.00
sequence load elapsed time	0.00	0.00
DB time	807.89	
background elapsed time	139.93	
background cpu time	0.82	



## Practice 7-1: Using AWR-Based Tools (continued)

- d) Scroll down to the Buffer Wait Statistics section. (Or, select Back to Top > Wait Statistics > Buffer Wait Statistics.)

What is the class of waits? \_\_\_\_\_ .  
 (“Waits on data blocks” is the expected answer.)

Buffer Wait Statistics			
<ul style="list-style-type: none"> <li>ordered by wait time desc, waits desc</li> </ul>			
Class	Waits	Total Wait Time (s)	Avg Time (ms)
data block	1,374	396	288
undo header	9	3	364
segment header	19	1	77

- e) Scroll down to “Segments by Buffer Busy Waits.” (Or, select Back to Top > Segment Statistics > Segments by Buffer Busy Waits.)

Which segment has the most buffer busy waits? \_\_\_\_\_ (SPCT)

Segments by Buffer Busy Waits						
<ul style="list-style-type: none"> <li>% of Capture shows % of Buffer Busy Waits for each top segment compared</li> <li>with total Buffer Busy Waits for all segments captured by the Snapshot</li> </ul>						
Owner	Tablespace Name	Object Name	Subobject Name	Obj. Type	Buffer Busy Waits	% of Capture
SPC	TBSSPC	SPCT		TABLE	1,384	99.35
SYS	SYSTEM	AUD\$		TABLE	9	0.65

- f) At this point, you have determined that there are a large number of buffer busy waits on a single segment.

7. View the ADDM report that was generated when you created the snapshot in step 4.

Step	Page	Action		
a	Database home	Click Advisor Central in the Related Links section.		
b	Advisor Central	Enter search values:	Field	Value
			Advisory Type	ADDM

## Practice 7-1: Using AWR-Based Tools (continued)

			Advisor Runs	Last 24 Hours
		Click Go.		

- c) Find and select the ADDM run that covers the snapshots you recorded in steps 2 and 4 by checking the description field of each ADDM report. The description will be in the “ADDM auto run: snapshots [177, 178], instance 1, database id 1159023676” format. Substitute your start and end snapshot numbers. (The latest ADDM report should be the one that meets the criteria.) Click View Result.
  - d) On the Automatic Database Diagnostic Monitor (ADDM) page, click the “Buffer Busy” finding. If more than one finding with this name appears on the page, check all of them.
  - e) On the Performance Finding Details page, you should see that ADDM recommends reorganizing the SPCT table to use Automatic Segment Space Management. Although you are able to infer this by looking at either a Statspack report or an AWR report, ADDM automatically tells you what to do. This recommendation may not appear. ADDM considers many factors before it gives a recommendation. A delay in creating the snapshot after the workload finishes is one reason that the recommendation may not appear.
8. View the ADDM findings from the Database home page.
    - a) Click the Database tab to return to the Database Instance home page. The latest ADDM report may not be visible until a few automatic refreshes take place. After a few automatic refreshes, you can see the latest performance analysis reported on the Database home page.
    - b) In the Performance Analysis section, click the “Buffer busy” finding. This shows the same page as in step 7c.
  9. Implement the recommendation.
    - a) To implement the suggested recommendation, execute the lab\_07\_01\_09.sql script. This script moves the table used in the workload to a tablespace by using Automatic Segment Space Management. You can locate the lab\_07\_01\_09.sql script in your \$HOME/labs directory.

```
$  
$ sqlplus / as sysdba  
SQL> @lab_07_01_09.sql  
SQL>  
SQL> drop tablespace tbsspc  
2 including contents and datafiles;
```

**Practice 7-1: Using AWR-Based Tools (continued)**

Tablespace dropped.

SQL>

```
SQL> CREATE SMALLFILE TABLESPACE "TBSSPC"
  2 DATAFILE 'tbsspc1.dbf' SIZE 50M
  3 AUTOEXTEND ON NEXT 10M MAXSIZE 200M
  4 LOGGING
  5 EXTENT MANAGEMENT LOCAL
  6 SEGMENT SPACE MANAGEMENT AUTO;
```

Tablespace created.

SQL>

SQL> connect spc/spc

Connected.

SQL>

SQL> drop table spct purge;

drop table spct purge

\*

ERROR at line 1:

ORA-00942: table or view does not exist

SQL> create table spct(id number, name varchar2(2000))

2 tablespace tbsspc;

Table created.

SQL>

SQL> exec DBMS\_STATS.GATHER\_TABLE\_STATS(-

> ownname=>'SPC', tabname=>'SPCT',-

> estimate\_percent=>DBMS\_STATS.AUTO\_SAMPLE\_SIZE);

PL/SQL procedure successfully completed.

SQL>

SQL> exit;

Disconnected from Oracle Database 11g Enterprise Edition

Release 11.1.0.6.0 - Production

With the Partitioning, Oracle Label Security, OLAP, Data

Mining

and Real Application Testing options

\$

10. Generate the workload again to observe the differences that the recommendation made. Execute the **./workgen 7 1** script to start generating the workload for this practice again.

\$ **cd \$HOME/workshops**

\$ **./workgen 7 1**

PL/SQL procedure successfully completed

## Practice 7-1: Using AWR-Based Tools (continued)

```
PL/SQL procedure successfully completed
PL/SQL procedure successfully completed
...
Load is finished
$
```

11. When the script finishes, take a third AWR snapshot. Record the snapshot number: \_\_\_\_\_
12. Generate a Compare Periods Report comparing the periods between the three snapshots.
  - a) You can use the `awrddrpt.sql` script located in your `$ORACLE_HOME/rdbms/admin` directory to generate the diff-diff report. The solution in this document uses Enterprise Manager. On the Snapshots page, select Compare Periods from the Actions drop-down list. Make sure that your first AWR snapshot is selected. (This snapshot number was recorded in step 2g.) Then, click Go.
  - b) On the Compare Periods: First Period End page, select your second AWR snapshot (the snapshot number recorded in step 4) and click Next.
  - c) On the Compare Periods: Second Period Start page, select your second AWR snapshot (the snapshot number recorded in step 4) and click Next.
  - d) On the Compare Periods: Second Period End page, select your last AWR snapshot (the snapshot number recorded in step 11) and click Next.
  - e) On the Compare Periods: Review page, click Finish.
13. Review the Compare Periods Report.
  - a) On the Compare Periods: Results page, select Per Transaction from the View Data drop-down list on the General tabbed page.
  - b) You can see the performance differences between the two analyzed periods. Click the Report tab.
  - c) The Processing page appears. Wait until the report is generated.
  - d) On the Report tabbed page, you can now see the Workload Repository Compare Period Report.
  - e) Scroll down to the Top Timed Events section.  
What is the %DB time for buffer busy waits in the first period? \_\_\_\_\_  
What is the %DB time for buffer busy waits in the second period?  
\_\_\_\_\_
  - f) In the Report Details section, click Wait Stats.
  - g) In the Wait Stats section, click Buffer Wait Statistics.

## Practice 7-1: Using AWR-Based Tools (continued)

- h) In the Buffer Wait Statistics report:  
 What is the wait time % of DB time for data blocks for the first period?  
 \_\_\_\_\_  
 What is the wait time % of DB time for data blocks for the second period?  
 \_\_\_\_\_
- i) Click Back to Top to return to the Report Details section.
- j) In the Report Details section, click Segment Statistics.
- k) In the Segment Statistics section, click Top 5 Segments by Buffer Busy Waits.
- l) Examine the Top 5 Segments by Buffer Busy Waits report.  
 What is the number of buffer busy waits in the first period?  
 \_\_\_\_\_  
 What is the number of buffer busy waits in the second period?  
 \_\_\_\_\_

14. Generate the ASH Report from the Top Activity page for the period with the greatest load. You can use the `ashrpt.sql` script located in the `$ORACLE_HOME/rdbms/admin` directory to generate the corresponding ASH Report. The solution in this document uses Enterprise Manager.

Step	Page	Action
a	Database home	Click the Performance tab.
b	Performance	Note the time of the first peak in the Active Sessions Graph. Pick a 10-minute interval that brackets the peak.
		Click Run ASH Report.
c	Run ASH Report	Enter the Start time and End time from the times that you noted in the previous step.  Click Generate Report.
d	Processing	Wait until the report is generated.

15. View the ASH report.

- a) From the first section, you can get various general details from the report.
- b) Scroll down to the Top User Events section.  
 What is the top event? \_\_\_\_\_ (buffer busy waits is in the top events.)
- c) Scroll down to the Top SQL Command Types section.  
 What is the type of the top SQL statement? \_\_\_\_\_ (INSERT)

### **Practice 7-1: Using AWR-Based Tools (continued)**

- d) Scroll down to the Top SQL with Top Events section. What is the top SQL statement and what was the primary wait event for that statement?

What is the SQL ID? \_\_\_\_\_ (3csh3g3mjhmzh)

- e) Scroll down to the Activity Over Time section.  
How many periods are reported? \_\_\_\_\_ (11, maybe less, some machines show only 1 period, when the activity otherwise is very low.)  
What is the elapsed time of each period? \_\_\_\_\_ (This varies, usually 1 minute, except that the first and last periods may be shorter to allow periods to fall on minute boundaries.)  
Which periods show buffer busy waits? \_\_\_\_\_ (possibly none if buffer busy waits are not the top event)  
These answers may vary greatly.

16. Clean up this practice by running the `./cleanup 7 1` script.

```
$ ./cleanup 7 1

Tablespace dropped.

User dropped.

PL/SQL procedure successfully completed.
```

## Practices for Lesson 8

In this practice, you monitor applications with services.

## Practice 8-1: Using Services with a Single-Instance Oracle Database

The first step in creating a service configuration is to plan services for each application or application function that uses your database.

In this practice, you create the following configuration in the `orcl` database:

Service Name	Usage	Response Time (sec)– Warning/Critical
SERV1	Client service	0.4, 1.0

- 1) Use the `DBMS_SERVICE` package to create a service called `SERV1`. Then, make sure that you add your service name to your `tnsnames.ora` file.
  - a) The recommended method of adding a service name to the `tnsnames.ora` file is to use Net Manager. For this exercise, execute the `lab_08_01_01.sh` script. Review the `tnsnames.ora` file at `$ORACLE_HOME/network/admin` to confirm that the following lines are included. Substitute the output of the `hostname` command for `<hostname>` in the following screenshot:

```
SERV1 =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)
      (HOST = <hostname>.us.oracle.com)(PORT = 1521))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = SERV1.us.oracle.com)
    )
  )
```

```
$ cd /home/oracle/labs
$ ./lab_08_01_01.sh
edrsr3p1
$
```

- 2) Use the `DBMS_SERVICE.CREATE_SERVICE` procedure to create a service.

```
$ sqlplus / as sysdba

SQL> EXEC -
DBMS_SERVICE.CREATE_SERVICE('SERV1','SERV1.us.oracle.com')

PL/SQL procedure successfully completed.

SQL> exit;
```



## Practice 8-1: Using Services with a Single-Instance Oracle Database (continued)

- 3) After you have created your services, try connecting to your database by using your service name. What happens and why? \_\_\_\_\_

(You cannot connect by using your service because, although it is defined, it is not started on your instance. You can verify this by looking at the SERVICE\_NAMES initialization parameter, and by looking at the services known to the listener.)

```
$ sqlplus / as sysdba

SQL> show parameter service

NAME                                TYPE        VALUE
-----
service_names                       string      orcl.oracle.com

SQL> connect system@SERV1
Enter password: oracle_4U /* password is not displayed */
ERROR:
ORA-12514: TNS:listener does not currently know of service
requested in connect
descriptor

Warning: You are no longer connected to ORACLE.
SQL> exit

$ lsnrctl services

Connecting to
(DESCRIPTION=(ADDRESS=(PROTOCOL=IPC)(KEY=EXTPROC1521)))
Services Summary...
Service "orcl" has 1 instance(s).
  Instance "orcl", status READY, has 1 handler(s) for this
service...
    Handler(s):
      "DEDICATED" established:1465 refused:0 state:ready
      LOCAL SERVER
Service "orcl.us.oracle.com" has 1 instance(s).
  Instance "orcl", status READY, has 1 handler(s) for this
service...
    Handler(s):
      "DEDICATED" established:1465 refused:0 state:ready
      LOCAL SERVER
Service "orclXDB" has 1 instance(s).
  Instance "orcl", status READY, has 1 handler(s) for this
service...
    Handler(s):
      "D000" established:0 refused:0 current:0 max:1022
state:ready
```

## Practice 8-1: Using Services with a Single-Instance Oracle Database (continued)

```
DISPATCHER <machine: edrsr9p0.us.oracle.com, pid:
3998>

(ADDRESS=(PROTOCOL=tcp)(HOST=edrsr9p0.us.oracle.com)(PORT=3277
4))
Service "orcl_XPT" has 1 instance(s).
Instance "orcl", status READY, has 1 handler(s) for this
service...
Handler(s):
  "DEDICATED" established:1465 refused:0 state:ready
  LOCAL SERVER
The command completed successfully
$
```

- 4) How would you make sure that you can connect using your service? Attempt your solution, and connect to your instance using your service.
- a) **Answer:** You must start the `SERV1` service on your instance.
- Note:** If the service does not appear in the `lsnrctl services` listing immediately, run the `ALTER SYSTEM REGISTER;` command, and then retry the connection by running the `connect system/oracle@SERV1` command.

```
$ sqlplus / as sysdba

SQL*Plus: Release 11.1.0.6.0 - Production on Thu Mar 27
09:10:31 2008

Copyright (c) 1982, 2007, Oracle. All rights reserved.

Connected to:
Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 -
Production
With the Partitioning, Oracle Label Security, OLAP, Data
Mining
and Real Application Testing options

SQL> show parameter service

NAME                                TYPE                                VALUE
-----                                -                                -
service_names                        string                             orcl.us.oracle.com

SQL> EXEC DBMS_SERVICE.START_SERVICE('SERV1');

PL/SQL procedure successfully completed.

SQL> show parameter service

NAME                                TYPE                                VALUE
-----                                -                                -
```

## Practice 8-1: Using Services with a Single-Instance Oracle Database (continued)

```
-----
service_names      string      orcl.us.oracle.com, SERV1.us.o
                        racle.com

SQL> host lsnrctl services

LSNRCTL for Linux: Version 11.1.0.6.0 - Production on 27-MAR-
2008 09:13:39

Copyright (c) 1991, 2007, Oracle. All rights reserved.

Connecting to
(DESCRIPTION=(ADDRESS=(PROTOCOL=IPC)(KEY=EXTPROC1521)))
Services Summary...
Service "SERV1.us.oracle.com" has 1 instance(s).
  Instance "orcl", status READY, has 1 handler(s) for this
service...
  Handler(s):
    "DEDICATED" established:1 refused:0 state:ready
    LOCAL SERVER
Service "orcl" has 1 instance(s).
  Instance "orcl", status READY, has 1 handler(s) for this
service...
  Handler(s):
    "DEDICATED" established:1 refused:0 state:ready
    LOCAL SERVER
Service "orcl.us.oracle.com" has 1 instance(s).
  Instance "orcl", status READY, has 1 handler(s) for this
service...
  Handler(s):
    "DEDICATED" established:1 refused:0 state:ready
    LOCAL SERVER
Service "orclXDB" has 1 instance(s).
  Instance "orcl", status READY, has 1 handler(s) for this
service...
  Handler(s):
    "D000" established:0 refused:0 current:0 max:1022
state:ready
    DISPATCHER <machine: edrsr9p0.us.oracle.com, pid:
3998>

(ADDRESS=(PROTOCOL=tcp)(HOST=edrsr9p0.us.oracle.com)(PORT=3277
4))
Service "orcl_XPT" has 1 instance(s).
  Instance "orcl", status READY, has 1 handler(s) for this
service...
  Handler(s):
    "DEDICATED" established:1 refused:0 state:ready
    LOCAL SERVER
The command completed successfully
```

## Practice 8-1: Using Services with a Single-Instance Oracle Database (continued)

```
$ sqlplus /nolog
SQL> connect SYSTEM@SERV1
Enter password: oracle_4U /* Password is not displayed */
Connected.
SQL> exit
Disconnected from Oracle Database 11g Enterprise Edition
Release 11.1.0.6.0 - Production
With the Partitioning, Oracle Label Security, OLAP, Data
Mining
and Real Application Testing options
```

- 5) From the /home/oracle/workshops directory, execute the ./workgen 8 1 script. This workload will run until stopped; continue to the next step. This script starts three workloads, each connecting with a different service name: SERV1, ORCL, and SYS\$.

```
$ cd /home/oracle/workshops
$ ./workgen 8 1
$ ps
  PID TTY          TIME CMD
 2386 pts/4        00:00:00 bash
 23752 pts/4        00:00:00 workgen
 23753 pts/4        00:00:00 workgen
 23754 pts/4        00:00:00 wksh_w81_orcl.s
 23756 pts/4        00:00:00 sqlplus
 23781 pts/4        00:00:00 insert_orders.s
 23794 pts/4        00:00:00 sqlplus
 23797 pts/4        00:00:00 update_orders.s
 23805 pts/4        00:00:00 delete_orders.s
 23811 pts/4        00:00:00 insert_orders.s
 23816 pts/4        00:00:00 sleep
 23818 pts/4        00:00:00 update_orders.s
 23823 pts/4        00:00:00 sleep
 23824 pts/4        00:00:00 delete_orders.s
 23829 pts/4        00:00:00 sleep
 23834 pts/4        00:00:00 sleep
 23835 pts/4        00:00:00 insert_orders.s
 23843 pts/4        00:00:00 sleep
 23844 pts/4        00:00:00 sleep
 23846 pts/4        00:00:00 update_orders.s
 23847 pts/4        00:00:00 sleep
 23851 pts/4        00:00:00 sleep
 23852 pts/4        00:00:00 sqlplus
 23855 pts/4        00:00:00 ps
$
```

## Practice 8-1: Using Services with a Single-Instance Oracle Database (continued)

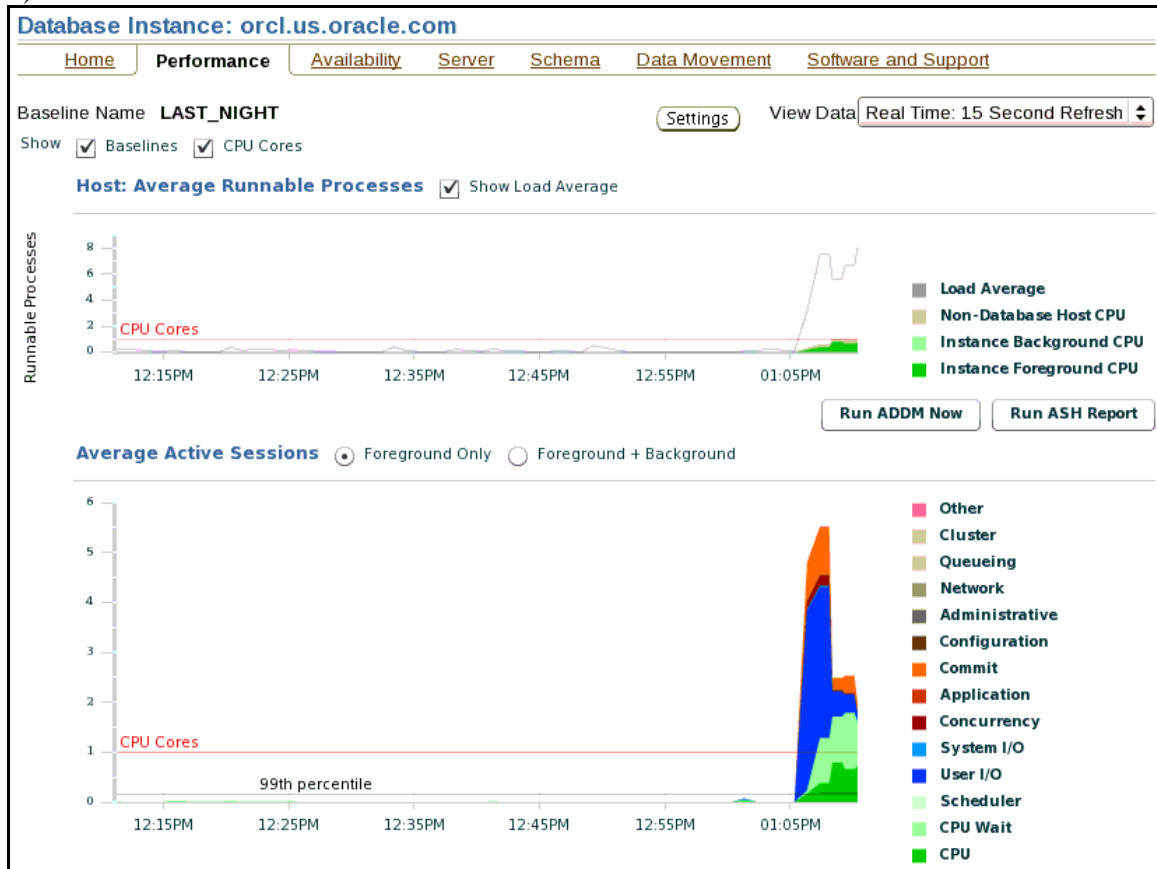
- 6) After the execution starts, access the EM Top Consumers page from the Performance tabbed page, and observe that SERV1 is using some resources. Also, check the statistics on your service with V\$SERVICE\_STATS from a SQL\*Plus session connected as SYSDBA.

**Note:** Screenshots of the steps follow the table.

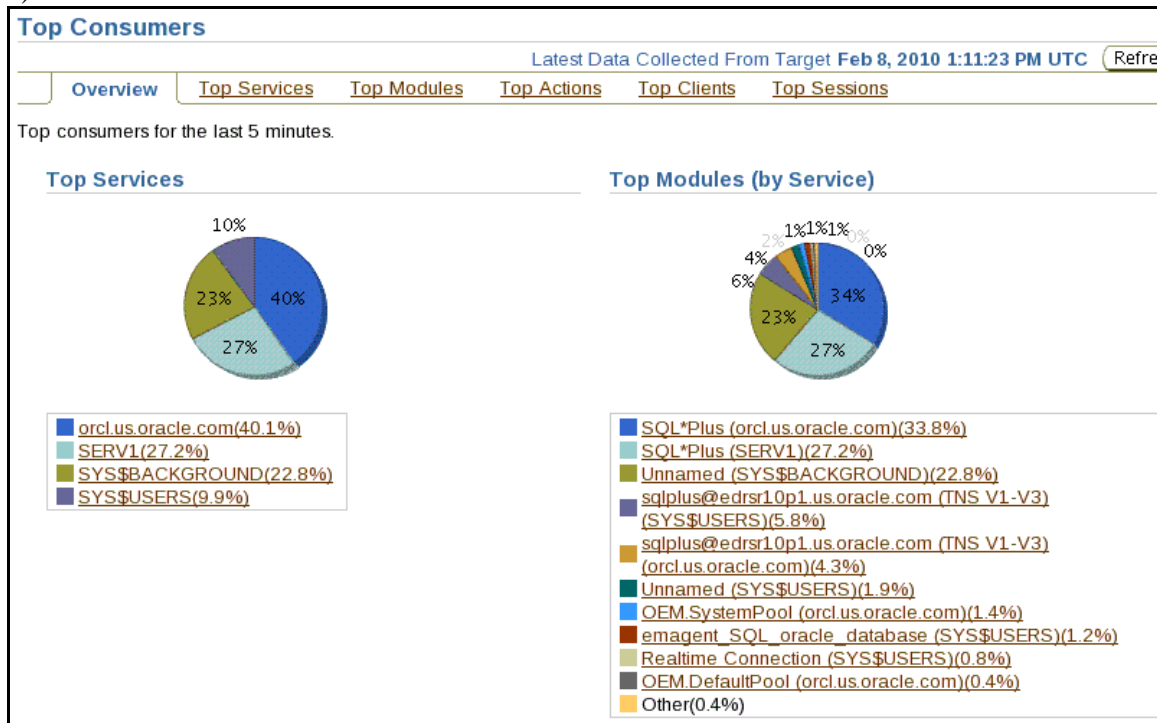
Step	Page	Action	Observation
a	Database Instance home	Click the Performance tab.	
b	Performance (See the example screenshot that follows the table.)		On the Performance page, wait at this point until the Average Active Sessions graph shows a significant rise.
		Click Top Consumers in the Additional Monitoring Links section.	
c	Top Consumers Overview (See the example screenshot, which follows the table.)	Click Refresh.	The names and number of services listed in the Top Services graph depend on the number and type of connections to the database. The network service name of each connection is recorded as a separate service. As a result, all the connections made without a service name will be aggregated, as will all the connections made as SERV1.
		Click the Top Services tab.	
d	Top Services	Click the SERV1 link.	
e	Service:SERV1 Modules tab	Click the Statistics tab.	
f	Service:SERV1 Statistics tab		Notice that the statistics are the same as the AWR report, but aggregated by service.

## Practice 8-1: Using Services with a Single-Instance Oracle Database (continued)

b)



c)

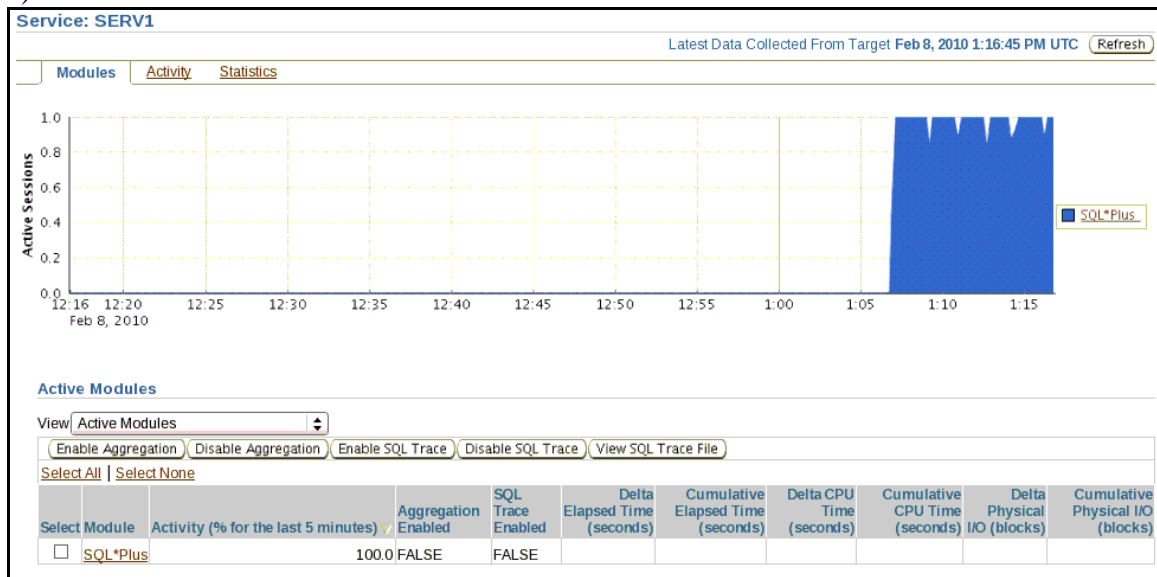


## Practice 8-1: Using Services with a Single-Instance Oracle Database (continued)

d)

Top Consumers									
Latest Data Collected From Target Feb 8, 2010 1:13:23 PM UTC									
<a href="#">Overview</a> <a href="#">Top Services</a> <a href="#">Top Modules</a> <a href="#">Top Actions</a> <a href="#">Top Clients</a> <a href="#">Top Sessions</a>									
View <input type="text" value="Active Services"/>									
<input type="button" value="Enable SQL Trace"/> <input type="button" value="Disable SQL Trace"/> <input type="button" value="View SQL Trace File"/>									
<a href="#">Select All</a>   <a href="#">Select None</a>									
Select	Service	Activity (% for the last 5 minutes)	SQL Trace Enabled	Delta Elapsed Time (seconds)	Cumulative Elapsed Time (seconds)	Delta CPU Time (seconds)	Cumulative CPU Time (seconds)	Delta Physical I/O (blocks)	Cumulative Physical I/O (blocks)
<input type="checkbox"/>	SERV1	42.6	FALSE	16	388	8	189	32	
<input type="checkbox"/>	orcl.us.oracle.com	32.1	FALSE	11	1341	3	430	352	
<input type="checkbox"/>	SYS\$BACKGROUND	20.6	FALSE	0	0	0	0	560	
<input type="checkbox"/>	SYS\$USERS	4.7	FALSE	0	3760	0	1720	24	

e)



## Practice 8-1: Using Services with a Single-Instance Oracle Database (continued)

f)

Database Instance: [ord.us.oracle.com](#) > [Top Consumers](#) >

Logged in As SYS

View Data

Real Time: 15 Second Refresh

Service: SERV1

Latest Data Collected From Target Feb 8, 2010 1:18:00 PM UTC

Refresh

Modules

Activity

Statistics

Previous

1-25 of 28

Next 3

Name	Delta Value	Cumulative Value
logons cumulative	0	9
user calls	0	170
DB time	13801585	656699136
DB CPU	7770819	330949696
parse count (total)	0	197
parse time elapsed	0	3502068
execute count	0	226
sql execute elapsed time	13801086	653518912
opened cursors cumulative	0	230
session logical reads	1426	43289
physical reads	1135	38112
physical writes	0	0
redo size	0	46684
user commits	0	0

- 7) Stop the running workload by removing the \$HOME/workshops/runload file.

```
$ rm $HOME/workshops/runload
```

- 8) Set alert thresholds for your SERV1 service by using Database Control. Specify a warning threshold of 40,000,000 microseconds and a critical threshold of 100,000,000 microseconds.

Step	Page	Action
a	Database home	Click the Metric and Policy Settings link in the Related Links section.
b	Metric and Policy Settings	Select All Metrics in View. (The page will refresh.)
		Click the multiedit icon (the group of pencils) for the “Service Response Time (per user call) (microseconds)” metric.



## Practice 8-1: Using Services with a Single-Instance Oracle Database (continued)

c	Edit Advanced Settings: Service Response Time (per user call) (microseconds)	Click Add.		
d	Specify Multiple Thresholds: Service Response Time (per user call) (microseconds)	Specify the threshold values:	<b>Fields</b>	<b>Values</b>
			Service Name	SERV1
			Warning Threshold	40000000
			Critical Threshold	100000000
		Click Continue.		
e	Metric and Policy Settings	Click OK.		
f	Confirmation	Click OK.		

9) Verify the metric settings.

Step	Page	Action
a	Database Control home	Click the All Metrics link in the Related Links section.
b	All Metrics	Expand Database Services.
		Click “Service Response Time (per user call) (microseconds).”
c	Service Response Time (per user call) (microseconds)	Click the SERV1 link in the Service Name column.
d	Service Response Time (per user call) (microseconds): Service Name SERV1	Select Real Time: 30 Second Refresh from the View Data.

10) From your terminal emulator session, execute the  
`/home/oracle/labs/lab_08_01_10.sh` script. This script executes a workload under the SERV1 service. Observe the Metric Value graph on the Service Response Time (per user call) (microseconds): Service Name SERV1 page.

- Execute the script. It takes a while to complete. Do not wait until it completes. Look at the corresponding threshold history by using Database Control.

## Practice 8-1: Using Services with a Single-Instance Oracle Database (continued)

```
$ cd $HOME/labs
$ ./lab_08_01_10.sh
```

11) While the workload is executing, observe the Service Response Time graph for the SERV1 service. You may have to wait a couple of minutes for the graph to respond.

Step	Page	Action	Answer
a	Service Response Time (per user call) (microseconds): Service Name SERV1	Monitor the Metric Value graph.	The graph shows the metric for the SERV1 service.
		What conclusion can you draw?	Services can be monitored individually.
		When an Alert is shown, navigate to the home page. Click the Database tab.	
b	Database Instance home	View Alerts.	An alert for the Service Response Time for the SERV1 service is present. See the screenshot that follows the table.

b)

Alerts					
Category		All	Go	Critical 0	Warning 3
Severity	Category	Name	Impact	Message	Alert Triggered
	User Audit	Audited User		User SYS logged on from edrsr10p1.us.oracle.com.	Feb 8, 2010 6:36:10 AM
	Database Services	Service Response Time (per user call) (microseconds)		Metrics "Elapsed Time Per User Call" is at 59289022 for service "SERV1"	Feb 8, 2010 1:33:12 PM
	Invalid Objects by Schema	Owner's Invalid Object Count		6 object(s) are invalid in the OE schema.	Jan 27, 2010 11:04:11 AM

## ***Practice 8-1: Using Services with a Single-Instance Oracle Database (continued)***

12) Use Database Control to remove the thresholds that you specified during this practice.

Step	Page	Action
a	Database Control home	Click Metric and Policy Settings in the Related Links section.
b	Metrics and Policies	Click the multiedit icon on the right for Service Response Time (per user call) (microseconds).
c	Edit Advanced Settings: Service Response Time (per user call) (microseconds)	Select SERV1.
		Click Remove.
		Click Continue.
d	Metrics and Policies Settings	Click OK.
e	Confirmation	Click OK.

## Practice 8-2: Tracing Services in a Single-Instance Oracle Environment

Unless specified differently, log in as SYSDBA in your Database Control Console.

- 1) Execute the `lab_08_02_01.sh` script to remove all the trace files already generated in your `$ORACLE_BASE/diag/rdbms/orcl/orcl/trace` directory.

```
$ cd $HOME/labs
$ ./lab_08_02_01.sh

trace_dir=$ORACLE_BASE/diag/rdbms/orcl/orcl/trace

mv $trace_dir $trace_dir.old
mkdir $trace_dir
mv $trace_dir.old/alert_orcl.log $trace_dir/
rm -rf $ORACLE_BASE/diag/rdbms/orcl/orcl/trace.old

$
```

- 2) Execute the `start_servwork.sh` script to start four sessions by using the `SERV1` service.

```
$ ./start_servwork.sh
Started stream with pid=7904
Started stream with pid=7905
Started stream with pid=7906
Started stream with pid=7907
$
PL/SQL procedure successfully completed.

PL/SQL procedure successfully completed.

PL/SQL procedure successfully completed.

PL/SQL procedure successfully completed.

PL/SQL procedure successfully completed.

PL/SQL procedure successfully completed.

PL/SQL procedure successfully completed.

PL/SQL procedure successfully completed.

$
```

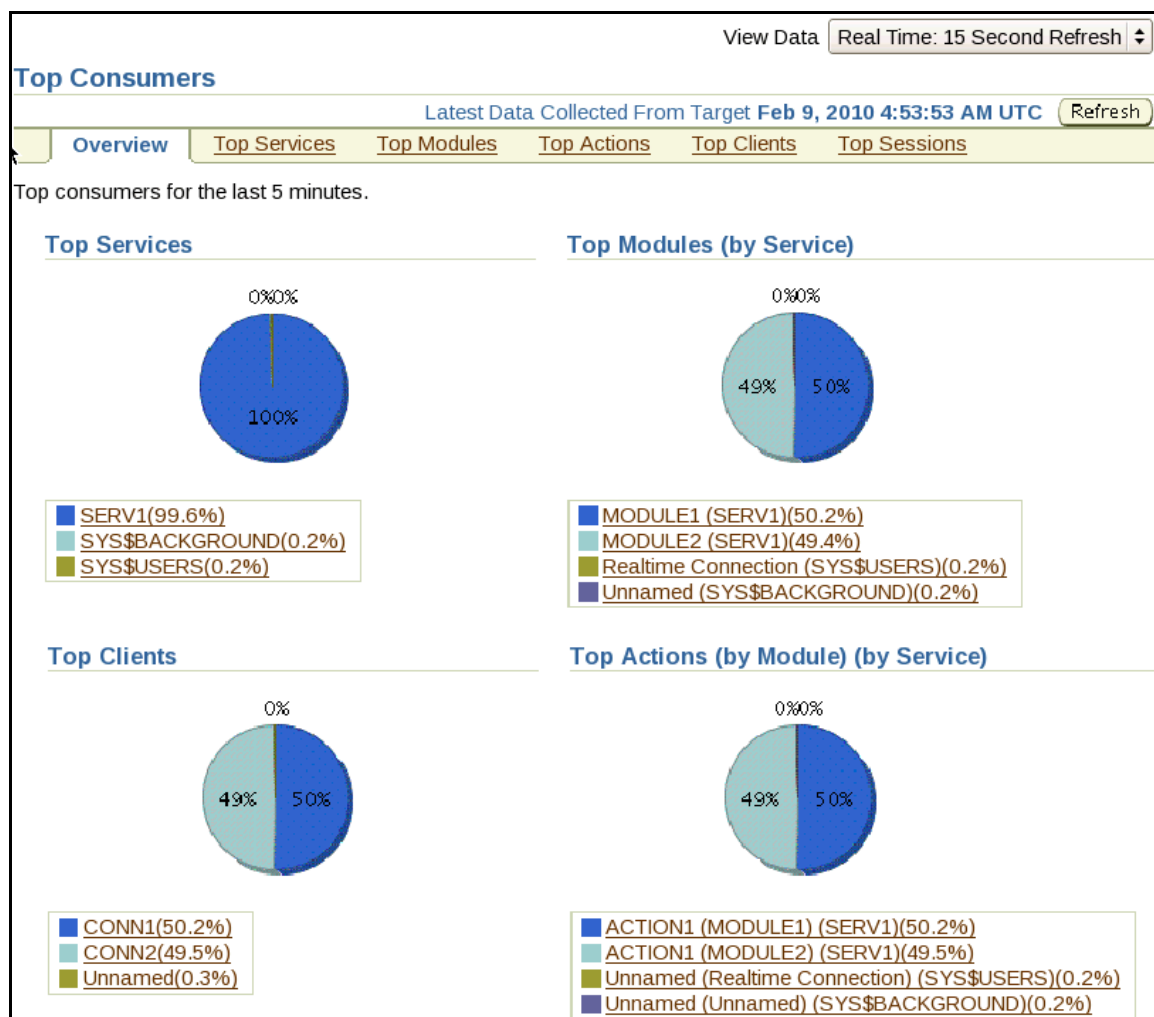
## ***Practice 8-2: Tracing Services in a Single-Instance Oracle Environment (continued)***

- 3) Using Database Control, determine the list of services, modules, and actions that the workload is using.

Step	Page	Action/Observe
a	Database Control home	Click the Performance tab.
b	Performance	Click the Top Consumers link in the Additional Monitoring Links section.
c	Top Consumers (See the example screenshot, which follows the table.)	You should see that the current workload uses only the SERV1 service, and that half of the sessions are using the SERV1/MODULE1/ACTION1 action and the other half is using the SERV1/MODULE2/ACTION1 action.

## Practice 8-2: Tracing Services in a Single-Instance Oracle Environment (continued)

The following screenshot displays the Top Consumers page:



- 4) Using Database Control, enable statistics aggregation for both SERV1/MODULE1/ACTION1 and SERV1/MODULE2/ACTION1.

Step	Page	Action/Observation
a	Top Consumers	Click the Top Actions tab.
b	Top Actions	Select the SERV1 /MODULE1 /ACTION1 action.
		Click Enable Aggregation.
c	Top Actions	Select the SERV1 /MODULE2 /ACTION1 action.
		Click Enable Aggregation.

## Practice 8-2: Tracing Services in a Single-Instance Oracle Environment (continued)

d	Top Actions	You should see that additional statistics at the right of the window are now tracked for both actions (may require a refresh to see the additional statistics).
---	-------------	---

d)

Top Consumers										
Latest Data Collected From Target Feb 9, 2010 4:57:38 AM UTC <span>Refresh</span>										
<span>Overview</span> <span>Top Services</span> <span>Top Modules</span> <b><span>Top Actions</span></b> <span>Top Clients</span> <span>Top Sessions</span>										
View <span>Active Actions</span>										
<span>Enable Aggregation</span> <span>Disable Aggregation</span> <span>Enable SQL Trace</span> <span>Disable SQL Trace</span> <span>View SQL Trace File</span>										
<span>Select All</span>   <span>Select None</span>										
Select	Service	Module	Action	Activity (% for the last 5 minutes)	Aggregation Enabled	SQL Trace Enabled	Delta Elapsed Time (seconds)	Cumulative Elapsed Time (seconds)	Delta CPU Time (seconds)	Cumulative CPU Time (seconds)
<input checked="" type="checkbox"/>	SERV1	MODULE1	ACTION1	50.0	TRUE	FALSE	0	0	0	0
<input checked="" type="checkbox"/>	SERV1	MODULE2	ACTION1	50.0	TRUE	FALSE	0	0	0	0
<input type="checkbox"/>	SYS\$BACKGROUND	Unnamed	Unnamed	.1	FALSE	FALSE				

- 5) Using Database Control, enable tracing for both SERV1/MODULE1/ACTION1 and SERV1/MODULE2/ACTION1. Let the system generate the trace files for one minute, and then disable tracing for both SERV1/MODULE1/ACTION1 and SERV1/MODULE2/ACTION1. After this is done, determine the list of generated trace files.

Step	Page	Action
a	Top Actions	Select Actions with Aggregation Enable from View.
		Select the SERV1/MODULE1/ACTION1 action.
		Click Enable SQL Trace.
b	Enable SQL Trace	Click OK.
c	Top Actions	Select Actions with Aggregation Enable from View.
		Select the SERV1/MODULE2/ACTION1 action.
		Click Enable SQL Trace.
d	Enable SQL Trace	Click OK.

- 6) Find the trace files. Use the following command:

```
$ ls -lt $ORACLE_BASE/diag/rdbms/orcl/orcl/trace/*.trc
```

```
$ ls -lt $ORACLE_BASE/diag/rdbms/orcl/orcl/trace/*.trc
-rw-r----- 1 oracle oinstall 10617510 Mar 30 07:13 orcl_ora_3952.trc
-rw-r----- 1 oracle oinstall 10827320 Mar 30 07:13 orcl_ora_3954.trc
```

## Practice 8-2: Tracing Services in a Single-Instance Oracle Environment (continued)

```
-rw-r----- 1 oracle oinstall      1624 Mar 30 07:12 orcl_dbrm_1633.trc
-rw-r----- 1 oracle oinstall 271200169 Mar 30 07:12 orcl_ora_3950.trc
-rw-r----- 1 oracle oinstall 253132737 Mar 30 07:11 orcl_ora_3959.trc
-rw-r----- 1 oracle oinstall      910 Mar 30 07:01 orcl_m000_3272.trc
$
```

- 7) Stop your workload by executing the `stop_servwork.sh` script, and then start it again by executing the `start_servwork.sh` script. Then check whether the statistics for your enabled aggregation actions are increasing.

```
$ ./stop_servwork.sh
Killing stream with pid=3067
Killing stream with pid=3064
Killing stream with pid=3063
Killing stream with pid=3062
$
$ ./start_servwork.sh
Started stream with pid=8535
Started stream with pid=8536
Started stream with pid=8539
Started stream with pid=8540
$
PL/SQL procedure successfully completed.

PL/SQL procedure successfully completed.

PL/SQL procedure successfully completed.

PL/SQL procedure successfully completed.

PL/SQL procedure successfully completed.

PL/SQL procedure successfully completed.

PL/SQL procedure successfully completed.

PL/SQL procedure successfully completed.

$
```

- 8) Look again at your Top Actions page. You should see statistics refreshing for your two actions.
- 9) Using Database Control, disable statistic aggregations for both the actions.



## Practice 8-2: Tracing Services in a Single-Instance Oracle Environment (continued)

- a) From the Top Actions page, select both the actions, and click the Disable Aggregation button.

10) Stop the workload by using the `stop_servwork.sh` script.

```
$ ./stop_servwork.sh
Killing stream with pid=3944
Killing stream with pid=3942
Killing stream with pid=3941
Killing stream with pid=3940
$
```

11) Use the trace session utility and `tkprof` to interpret the generated trace files for the `SERV1` service and the `MODULE1` module.

```
$ cd $ORACLE_BASE/diag/rdbms/orcl/orcl/trace
$ trcsess output=serv1_module1.trc service=SERV1 *ora*.trc
$ tkprof serv1_module1.trc results.trc

TKPROF: Release 11.2.0.1.0 - Development on Tue Feb 9 05:08:10
2010

Copyright (c) 1982, 2009, Oracle and/or its affiliates. All
rights reserved.
```

12) View the output of `tkprof` with your favorite viewer or editor; the viewer `less` is shown.

```
$ less results.trc

TKPROF: Release 11.2.0.1.0 - Development on Tue Feb 9 05:08:10 2010

Copyright (c) 1982, 2009, Oracle and/or its affiliates. All rights
reserved.

Trace file: serv1_module1.trc
Sort options: default

*****
count      = number of times OCI procedure was executed
cpu        = cpu time in seconds executing
elapsed    = elapsed time in seconds executing
disk       = number of physical reads of buffers from disk
query      = number of buffers gotten for consistent read
current    = number of buffers gotten in current mode (usually for update)
rows       = number of rows processed by the fetch or execute call
*****

SQL ID: 7r0kgzntdn7sq
Plan Hash: 2945320129
SELECT COUNT(*)
FROM DBA_OBJECTS
```

## Practice 8-2: Tracing Services in a Single-Instance Oracle Environment (continued)

call	count	cpu	elapsed	disk	query	current	rows
Parse	2	0.01	0.02	0	0	0	0
Execute	1240	0.05	0.05	0	0	0	0
Fetch	1235	289.61	1601.80	1123850	2274870	0	1235
total	2477	289.68	1601.88	1123850	2274870	0	1235

Misses in library cache during parse: 1  
 Optimizer mode: ALL\_ROWS  
 Parsing user id: 122 (recursive depth: 1)

Elapsed times include waiting on following events:

Event waited on	Times Waited	Max. Wait	Total Waited
direct path read	34	1.12	12.85
kfk: async disk IO	141171	0.51	3.53
latch: cache buffers chains	17	0.51	5.13
asynch descriptor resize	690	0.00	0.00
latch free	5	0.50	1.80

\*\*\*\*\*

SQL ID: 1hb6a7lhd3d53  
 Plan Hash: 0  
 BEGIN dbms\_application\_info.set\_module('MODULE1','ACTION1'); END;

call	count	cpu	elapsed	disk	query	current	rows
Parse	0	0.00	0.00	0	0	0	0
Execute	2	0.00	0.00	0	0	0	2
Fetch	0	0.00	0.00	0	0	0	0
total	2	0.00	0.00	0	0	0	2

Misses in library cache during parse: 0  
 Optimizer mode: ALL\_ROWS  
 Parsing user id: 122  
 ...  
 (type q to exit)

13) Clean up the database environment. Run the lab\_08\_02\_13.sh script.

```

$ cd $HOME/labs
$ ./lab_08_02_13.sh

PL/SQL procedure successfully completed.

PL/SQL procedure successfully completed.
$
  
```

## Practices for Lesson 9

In this practice, you examine and compare the information provided by some built-in diagnostic tools. You start with a poorly performing SQL statement and examine the output of the various tools to try to determine the reasons for poor performance.

Practice 9-1: Using AUTOTRACE and DBMS\_XPLAN

Practice 9-2: Using the SQL TRACE facility

### **Practice 9-1: Using AUTOTRACE and DBMS\_XPLAN**

In this practice, you examine a bad SQL statement and gather information about it by using the SQL\*Plus AUTOTRACE command.

1. Create the PLUSTRACE role. To use the SQL\*Plus AUTOTRACE command, the PLUSTRACE role must be granted to the user that wishes to use AUTOTRACE.

```
$ sqlplus / as sysdba

SQL> @?/sqlplus/admin/plustrce.sql
SQL>
SQL> drop role plustrace;
drop role plustrace
      *
ERROR at line 1:
ORA-01919: role 'PLUSTRACE' does not exist

SQL> create role plustrace;

Role created.

SQL>
SQL> grant select on v_$sesstat to plustrace;

Grant succeeded.

SQL> grant select on v_$statname to plustrace;

Grant succeeded.

SQL> grant select on v_$mystat to plustrace;

Grant succeeded.

SQL> grant plustrace to dba with admin option;

Grant succeeded.

SQL>
SQL> set echo off
SQL>
```

2. Grant the PLUSTRACE role to the SH user.

```
SQL> grant plustrace to sh;

Grant succeeded.
```

**Practice 9-1: Using AUTOTRACE and DBMS\_XPLAN (continued)**

```
SQL> exit
```

3. The SQL statement has been isolated, and you can view it in the `bad_sql.sql` file. The script is in the `$HOME/labs/sqltune` directory. What can be determined by inspecting this statement? Each observation produces more specific questions.

```
$ cd $HOME/labs/sqltune
$ cat bad_sql.sql
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;
```

- a) This statement has a join between two tables: SALES and CUSTOMERS.

**Information:** These are large tables in the Sales History (SH) schema.

**Question:** Are the statistics up-to-date on these tables?

- b) The join column is `cust_id` in both tables.

**Questions:** Are there indexes on these columns?

Do the indexes have up-to-date statistics?

Are the indexes built properly?

- c) The selection predicate is `CUST_FIRST_NAME='Dina'`.

**Questions:** Does `CUST_FIRST_NAME` have an index?

- d) The ORDER BY clause may require a sort.

**Question:** Is the sort area in memory sized properly?

4. Determine the information provided by AUTOTRACE about the execution of this statement.

**Note:** The Explain Plan output has been reformatted for readability.

- a) Run the `bad_sql.sql` script with AUTOTRACE ON.

```
$ sqlplus /nolog

SQL> connect sh/sh
Connected.
SQL> set autotrace on
SQL> @bad_sql.sql

...
TIME_ID      QUANTITY_SOLD  AMOUNT_SOLD
-----
24-DEC-01          1          32.27
24-DEC-01          1          33.25
```

## Practice 9-1: Using AUTOTRACE and DBMS\_XPLAN (continued)

```
28-DEC-01          1          56.96
28-DEC-01          1          56.96
```

895 rows selected.

Execution Plan

-----  
Plan hash value: 1897253553

```
-----
| Id | Operation                | Name          | Rows  | Bytes | TempSpc | Cost (%CPU)| Time     | Pstart | Pstop |
-----
|  0 | SELECT STATEMENT         |               |  5557 | 179K |          |    948   (2)| 00:00:12 |        |        |
|  1 |   SORT ORDER BY          |               |  5557 | 179K |  272K    |    948   (2)| 00:00:12 |        |        |
| * 2 |    HASH JOIN              |               |  5557 | 179K |          |    897   (2)| 00:00:11 |        |        |
| * 3 |     TABLE ACCESS FULL    | CUSTOMERS     |    43 |  516 |          |    405   (1)| 00:00:05 |        |        |
|  4 |      PARTITION RANGE ALL  |               |  918K |  18M |          |    498   (2)| 00:00:06 |    1   |    28 |
|  5 |       TABLE ACCESS FULL  | SALES         |  918K |  18M |          |    498   (2)| 00:00:06 |    1   |    28 |
-----
```

Predicate Information (identified by operation id):

```
-----
2 - access("C"."CUST_ID"="S"."CUST_ID")
3 - filter("CUST_FIRST_NAME"='Dina')
```

Statistics

```
-----
13701 recursive calls
      0 db block gets
  6083 consistent gets
  3213 physical reads
      0 redo size
23039 bytes sent via SQL*Net to client
  1068 bytes received via SQL*Net from client
     61 SQL*Net roundtrips to/from client
    103 sorts (memory)
      0 sorts (disk)
    895 rows processed
```

SQL> **set autotrace off**

b) What objects are accessed? \_\_\_\_\_

(The SALES and CUSTOMERS tables)

**Practice 9-1: Using AUTOTRACE and DBMS\_XPLAN (continued)**

- c) What object is accessed first? \_\_\_\_\_  
(The CUSTOMERS table is read into a hash table.)
- d) Which object is partitioned? \_\_\_\_\_  
(The SALES table is partitioned.)
- e) How many partitions are accessed? \_\_\_\_\_  
(All 28 partitions)
- f) How many rows are returned (Cardinality)? \_\_\_\_\_  
(895 rows)
- g) How many rows does the execution plan expect? \_\_\_\_\_  
(5557 rows)
- h) What is the cost? \_\_\_\_\_  
(948, may vary due to specific hardware)
- i) Are any indexes used? \_\_\_\_\_  
(No)
- j) How much sort space is expected? \_\_\_\_\_  
(179 KB)
- k) How many consistent reads (logical reads) were done? \_\_\_\_\_  
(6083)
- l) How many physical reads were done?  
\_\_\_\_\_  
(6120, may vary due to the preceding activity)
- m) Did the sort spill to disk? \_\_\_\_\_  
(No, sorts, disk was 0)
5. Use DBMS\_XPLAN.DISPLAY to just get the explain plan for the SQL statement. The command requires that the statement be explained with the EXPLAIN PLAN command; then, the plan table is displayed. The terminal window must be resized and set for at least a 108-character width to display properly.
- a) Generate an explain plan.

```
SQL> EXPLAIN PLAN FOR
2> select time_id, QUANTITY_SOLD, AMOUNT_SOLD
3>   from sales s, customers c
4>   where c.cust_id = s.cust_id and CUST_FIRST_NAME='Dina'
5>   order by time_id;

Explained
```

## Practice 9-1: Using AUTOTRACE and DBMS\_XPLAN (continued)

- b) Display the plan table. Note that this is the same explain plan that you received from the AUTOTRACE command.

```
SQL> SET LINESIZE 130
SQL> SET PAGESIZE 0
SQL> SELECT * FROM table(DBMS_XPLAN.DISPLAY);
Plan hash value: 1897253553
```

Id	Operation	Name	Rows	Bytes	TempSpc	Cost (%CPU)	Time	Pstart	Pstop
0	SELECT STATEMENT		5557	179K		948 (2)	00:00:12		
1	SORT ORDER BY		5557	179K	272K	948 (2)	00:00:12		
* 2	HASH JOIN		5557	179K		897 (2)	00:00:11		
* 3	TABLE ACCESS FULL	CUSTOMERS	43	516		405 (1)	00:00:06		
4	PARTITION RANGE ALL		918K	18M		489 (2)	00:00:06	1	28
5	TABLE ACCESS FULL	SALES	918K	18M		489 (2)	00:00:06	1	28

```
Predicate Information (identified by operation id):
-----
```

```
2 - access("C"."CUST_ID"="S"."CUST_ID")
3 - filter("CUST_FIRST_NAME"='Dina')
```

```
18 rows selected.
```



## Practice 9-2: Using the SQL TRACE Facility

In this practice, you trace a bad SQL statement and format the trace file with the tkprof utility.

1. Start a session to trace the SQL statement in the `bad_sql.sql` script as the SH user. Set the `TRACEFILE_IDENTIFIER` parameter for your session. Add the "badsql" string to the trace file name.

```
ALTER SESSION SET TRACEFILE_IDENTIFIER="badsql";
```

```
$ sqlplus /nolog

SQL> connect sh/sh
Connected.
SQL>

SQL> ALTER SESSION SET TRACEFILE_IDENTIFIER="badsql";
```

2. Enable tracing in your session with the command, and then run the `bad_sql.sql` script from the `$HOME/labs/sqltune` directory.

```
EXECUTE DBMS_SESSION.SET_SQL_TRACE(TRUE);
```

```
SQL> EXECUTE DBMS_SESSION.SET_SQL_TRACE(TRUE);

PL/SQL procedure successfully completed.

SQL> @$HOME/labs/sqltune/bad_sql.sql

...
TIME_ID      QUANTITY_SOLD  AMOUNT_SOLD
-----
24-DEC-01           1          32.27
24-DEC-01           1          33.25
28-DEC-01           1          56.96
28-DEC-01           1          56.96

895 rows selected.

SQL> exit
```

3. Find and format the trace file. The trace file will be in the `$ORACLE_BASE/diag/rdbms/orcl/orcl/trace` directory with the `badsql` string embedded in the name. The `ls *badsql*.trc` command returns the name of the trace file. Use that name of the trace file in the `tkprof` command.

```
$ cd $ORACLE_BASE/diag/rdbms/orcl/orcl/trace
$ ls *badsql*.trc
orcl_ora_14804_badsql.trc
$ tkprof orcl_ora_14804_badsql.trc badsql.txt EXPLAIN=sh/sh
SYS=NO
```

## Practice 9-2: Using the SQL TRACE Facility (continued)

```
TKPROF: Release 11.2.0.1.0 - Development on Tue Feb 9 06:55:44
2010
```

```
Copyright (c) 1982, 2009, Oracle and/or its affiliates. All
rights reserved.
$
```

4. View the badsql.txt file. Locate the SQL statement.

```
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
```

Find the trace summary below the statement as shown in the following example.

The values shown are examples. The values you see may differ.

How much CPU time was used to fetch the records? \_\_\_\_\_(0.81)

How much elapsed time was used for the entire query? \_\_\_\_\_(0.89)

What was the number of physical reads (disk)? \_\_\_\_\_(3102)

What was the number of logical reads (query + current)? \_\_\_\_\_(3160)

```
$ less $ORACLE_BASE/diag/rdbms/orcl/orcl/trace/badsql.txt

TKPROF: Release 11.2.0.1.0 - Development on Tue Feb 9 06:55:44 2010

Copyright (c) 1982, 2009, Oracle and/or its affiliates. All rights reserved.

Trace file: orcl_ora_14804_badsql.trc
Sort options: default

*****
count      = number of times OCI procedure was executed
cpu        = cpu time in seconds executing
elapsed    = elapsed time in seconds executing
disk       = number of physical reads of buffers from disk
query      = number of buffers gotten for consistent read
current    = number of buffers gotten in current mode (usually for update)
rows       = number of rows processed by the fetch or execute call
*****

...

*****

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

call      count      cpu      elapsed      disk      query      current      rows
-----
Parse      1         0.00         0.00         0          0          0          0
Execute    1         0.00         0.00         0          0          0          0
Fetch     61         0.81         0.99       3088       3175          0       895
-----
total      63         0.82         1.00       3088       3175          0       895

Misses in library cache during parse: 1
```

**Practice 9-2: Using the SQL TRACE Facility (continued)**

```

Optimizer mode: ALL_ROWS
Parsing user id: 85 (SH)

Rows      Row Source Operation
-----
      895  SORT ORDER BY (cr=3175 pr=3088 pw=0 time=1341 us cost=948 size=183381
card=5557)
      895  HASH JOIN (cr=3175 pr=3088 pw=0 time=132695 us cost=897 size=183381
card=5557)
       65  TABLE ACCESS FULL CUSTOMERS (cr=1457 pr=1454 pw=0 time=64 us cost=405
size=516 card=43)
    918843  PARTITION RANGE ALL PARTITION: 1 28 (cr=1718 pr=1634 pw=0 time=2431046
us cost=489 size=19295703 ca
rd=918843)
    918843  TABLE ACCESS FULL SALES PARTITION: 1 28 (cr=1718 pr=1634 pw=0
time=998014 us cost=489 size=1929570
3 card=918843)

Rows      Execution Plan
-----
      0  SELECT STATEMENT  MODE: ALL_ROWS
      895  SORT (ORDER BY)
      895  HASH JOIN
       65  TABLE ACCESS  MODE: ANALYZED (FULL) OF 'CUSTOMERS' (TABLE)
    918843  PARTITION RANGE (ALL) PARTITION: START=1 STOP=28
    918843  TABLE ACCESS  MODE: ANALYZED (FULL) OF 'SALES' (TABLE)
           PARTITION: START=1 STOP=28

...

*****

OVERALL TOTALS FOR ALL NON-RECURSIVE STATEMENTS

call      count      cpu      elapsed      disk      query      current      rows
-----
Parse      1          0.00        0.00          0          0          0          0
Execute    2          0.00        0.01          0          0          0          1
Fetch     61          0.81        0.99        3088        3175          0        895
-----
total      64          0.82        1.01        3088        3175          0        896

Misses in library cache during parse: 1
Misses in library cache during execute: 1

OVERALL TOTALS FOR ALL RECURSIVE STATEMENTS

call      count      cpu      elapsed      disk      query      current      rows
-----
Parse     18          0.02        0.04          0          0          0          0
Execute   36          0.02        0.04          0          1          2          1
Fetch    133          0.00        0.04          8        110          0        348
-----
total    187          0.06        0.13          8        111          2        349

Misses in library cache during parse: 18
Misses in library cache during execute: 19

      3  user  SQL statements in session.
     35  internal SQL statements in session.
     38  SQL statements in session.
      1  statement EXPLAINED in this session.

```

## Practice 9-2: Using the SQL TRACE Facility (continued)

```
*****
Trace file: orcl_ora_9029_badsql.trc
Trace file compatibility: 11.1.0.7
Sort options: default

    1 session in tracefile.
    3 user SQL statements in trace file.
   35 internal SQL statements in trace file.
   38 SQL statements in trace file.
   21 unique SQL statements in trace file.
    1 SQL statements EXPLAINED using schema:
      SH.prof$plan_table
      Default table was used.
      Table was created.
      Table was dropped.
   536 lines in trace file.
    14 elapsed seconds in trace file.

(type q to exit)
```

## Practices for Lesson 10

In the following practices, you demonstrate various ways to influence the optimizer by changing the SQL:

Practice 10-1: Using Extended Statistics

Practice 10-2: Stale Statistics

For all practices in this lesson, the environment must be changed by executing the `./prepare 10 1` script from the `$HOME/workshops` directory. The script needs to be run only once.

## Practice 10-1: Capturing Extended Statistics

In this practice, you create extended statistics over a multicolumn set and an expression.

1. Run the `prepare 10 1` script from the `$HOME/workshops` directory if you have not already done so.

```
$ $HOME/workshops/prepare 10 1

System altered.
```

2. In the `SH.CUSTOMERS` table, the `CUST_CITY`, `CUST_STATE_PROVINCE`, and `COUNTRY_ID` columns are frequently used together as predicates. Because these columns are related, the combined selectivity is very different from the selectivity that the optimizer would calculate. The optimizer assumes that the columns are independent. To capture the statistics over this set of columns, there are two methods: create a concatenated index over these columns and collect the statistics over the index, or create a column group.

Create a column group to collect statistics over this column group without creating an index. Examine the `lab_10_01_01.sql` script in the `$HOME/labs/sqltune` directory, and then execute it.

```
$ cd $HOME/labs/sqltune
$ sqlplus /nolog

SQL> connect sh/sh
Connected.
SQL> set echo on
SQL> @lab_10_01_01.sql
SQL> SET SERVEROUTPUT ON
SQL>
SQL> exec dbms_stats.drop_extended_stats(-
  2   'sh','customers',-
  3   '(CUST_CITY,CUST_STATE_PROVINCE,COUNTRY_ID)');

*
ERROR at line 1:
ORA-20000: extension
"(CUST_CITY,CUST_STATE_PROVINCE,COUNTRY_ID)" does not
exist
ORA-06512: at "SYS.DBMS_STATS", line 6893
ORA-06512: at "SYS.DBMS_STATS", line 28235
ORA-06512: at line 1

SQL>
SQL> DECLARE
  2   cg_name  VARCHAR2(30);
  3   BEGIN
  4       cg_name := dbms_stats.create_extended_stats(
  5                   'SH','CUSTOMERS',
  6                   '(CUST_CITY,CUST_STATE_PROVINCE,COUNTRY_ID)');

```

**Practice 10-1: Capturing Extended Statistics (continued)**

```

7
8  dbms_output.put_line('column group name is: ' || cg_name);
9
10 dbms_stats.gather_table_stats('SH','CUSTOMERS',
11     method_opt=>'for all columns size 1,for columns
12     (CUST_CITY,CUST_STATE_PROVINCE,COUNTRY_ID) size 3');
13 END;
14 /
column group name is: SYS_STUMZ$C3AIHLPBROI#SKA58H_N
PL/SQL procedure successfully completed.
SQL>

```

3. The SELECT statement in lab\_10\_01\_02.sql gives distinct values of each column in the group.

```

SQL> @lab_10_01_02.sql
SQL> select count(distinct(cust_city)) cities,
2      count(distinct(CUST_STATE_PROVINCE)) states,
3      count(distinct(country_id)) countries
4  from customers;

```

CITIES	STATES	COUNTRIES
620	145	19

4. The SELECT statement in lab\_10\_01\_03.sql shows the selectivity that the default algorithm in the optimizer would find.

```

SQL> @lab_10_01_03.sql
SQL> select 1/( count(distinct(cust_city))) *
2          1/( count(distinct(CUST_STATE_PROVINCE))) *
3          1/( count(distinct(country_id)))
default_selectivity
4  from customers
5  /

```

DEFAULT_SELECTIVITY
5.8545E-07

5. The PL/SQL block in lab\_10\_01\_04.sql retrieves the column group name.

```

SQL> @lab_10_01_04.sql
SQL> declare
2      cg_name VARCHAR2(30);
3  BEGIN
4      cg_name := dbms_stats.show_extended_stats_name
5                  ('SH','CUSTOMERS',
6                  '(CUST_CITY,CUST_STATE_PROVINCE,COUNTRY_ID)');
7
8  dbms_output.put_line('column group name is: ' || cg_name);

```

**Practice 10-1: Capturing Extended Statistics (continued)**

```

9
10 END;
11 /
column group name is:SYS_STUMZ$C3AIHLPBROI#SKA58H_N

PL/SQL procedure successfully completed.

```

6. The lab\_10\_01\_05.sql script retrieves the statistics on the column group. NUM\_DISTINCT shows the number of distinct values for the three columns. The selectivity for the column group is 1/620 or 1.61E-3, rather than the 5.8E-7 that the default calculation would have given. The extended statistics help the optimizer make better choices, with more accurate selectivity when these columns are used together in a predicate.

```

SQL> @lab_10_01_05.sql
SQL> select e.extension col_group, t.num_distinct, t.histogram
2      from user_stat_extensions e, user_tab_col_statistics t
3      where e.extension_name=t.column_name
4      and t.table_name='CUSTOMERS';

COL_GROUP
-----
-----
NUM_DISTINCT HISTOGRAM
-----
("CUST_CITY", "CUST_STATE_PROVINCE", "COUNTRY_ID")
620 HEIGHT BALANCED

```

7. Tables often have columns that are not populated that are called “virtual columns.” They are calculated as part of the SELECT statement. When these virtual columns are used in a predicate, ORDER BY, or join conditions, the optimizer needs statistics to make fully informed decisions. The lab\_10\_01\_07.sql script creates a table with four columns and populates it. The id column contains 1,000 distinct numbers: 1 to 999. The second column contains only an X. The Cid column contains id as a character string. The Nid column contains (id\*100)+6, so that an id of 27 becomes the Nid of 2706.

```

SQL> @lab_10_01_07.sql
SQL>
SQL> DROP TABLE EXP_TEST PURGE;

Table dropped.

SQL>
SQL> Create table exp_test ( id Number(6,0),
2                          pad      varchar2 (20),
3                          Cid      VARCHAR(20),
4                          Nid      NUMBER(6,0));

```



**Practice 10-1: Capturing Extended Statistics (continued)**

Table created.

```
SQL>
SQL> Begin
  2  FOR i in 1..1000 LOOP
  3
  4  INSERT INTO exp_test values(
  5      i, 'X', TO_CHAR(i), i*100+6);
  6  END LOOP;
  7  END;
  8  /
```

PL/SQL procedure successfully completed.

8. When a column named in the predicate has a function applied to it, the index on the column cannot be used. A function-based index on the expression can be created to allow an index access to be used. The lab\_10\_01\_08 script creates a function-based index over the `(mod(id,10))` expression.

```
SQL> @lab_10_01_08.sql
SQL> set echo on
SQL>
SQL> CREATE index exp_mod on exp_test(mod(id,10));
```

Index created.

SQL>

9. With the lab\_10\_01\_09.sql script, create a virtual column over the `(mod(nid,10))` expression.

```
SQL> @lab_10_01_09.sql
SQL> set echo on
SQL>
SQL> select dbms_stats.create_extended_stats(
  2      'sh','exp_test','(mod(Nid,10))' ) from dual;

DBMS_STATS.CREATE_EXTENDED_STATS('SH','EXP_TEST','(MOD(NID,10))')
-----
SYS_STUFYQ8TH8YVE2JG8NLWCN9XTR
```

10. Gather statistics on the EXP\_TEST table. The virtual column named in step 9 is explicitly named in the method\_opt parameter.

```
SQL> @lab_10_01_10.sql
SQL> set echo on
SQL>
SQL> exec dbms_stats.gather_table_stats('sh','exp_test',-
```

## Practice 10-1: Capturing Extended Statistics (continued)

```
> method_opt=>'for all columns size 1 for columns  
(mod(Nid,10)) size 3');  
  
PL/SQL procedure successfully completed.  
  
SQL>
```

11. The extended statistics have been calculated for the EXP\_TEST table. The extended statistics have been defined for (MOD(nid,10)). What are the values of the extended statistics?

```
SQL> @lab_10_01_11.sql  
SQL> select e.extension col_group, t.num_distinct, t.histogram  
2      from user_stat_extensions e, user_tab_col_statistics t  
3      where e.extension_name=t.column_name  
4      and t.table_name='EXP_TEST';  
  
COL_GROUP  
-----  
-----  
NUM_DISTINCT HISTOGRAM  
-----  
  
(MOD("ID",10))  
          10 NONE  
  
(MOD("NID",10))  
          1 FREQUENCY  
  
SQL> exit
```

The extended statistics were calculated over the EXP\_MOD functional index that was created in step 8, as well as over the virtual column defined in step 9.

## Practice 10-2: Stale Statistics

In this practice, you see that statistics can influence the optimizer performance.

1. Create a table and run statistics on it and the associated indexes as the SH user.  
Use the `cr_badstats.sql` script in the `$HOME/labs/sqltune` directory to create the CTEST table and the associated indexes, and to set up the scenario. Statistics are gathered, and then enough rows are inserted into the CTEST table to render the statistics stale.

```
$ cd $HOME/labs/sqltune
$ sqlplus /nolog

SQL> connect sh/sh
Connected
SQL> @cr_badstats.sql
SQL> -- Create a table with indexes that will yield a
different
SQL> -- execution plan when statistics are gathered
SQL>
SQL> -- Create a table, indexes and gather statistics
SQL>
SQL> DROP SEQUENCE ctest_pk_seq;
DROP SEQUENCE ctest_pk_seq
          *
ERROR at line 1:
ORA-02289: sequence does not exist

SQL>
SQL> DROP TABLE CTEST PURGE;
DROP TABLE CTEST PURGE
          *
ERROR at line 1:
ORA-00942: table or view does not exist

SQL>
SQL> CREATE TABLE CTEST AS SELECT * FROM CUSTOMERS;

Table created.

SQL>
SQL> ALTER TABLE CTEST MODIFY (CUST_ID PRIMARY KEY);

Table altered.

SQL>
SQL> CREATE INDEX CTEST_LASTNAME ON CTEST (CUST_LAST_NAME);

Index created.

SQL>
```

**Practice 10-2: Stale Statistics (continued)**

```

SQL> CREATE INDEX CTEST_COUNTRY ON CTEST ( COUNTRY_ID);
Index created.

SQL>
SQL> EXECUTE DBMS_STATS.GATHER_TABLE_STATS('SH','CTEST');
PL/SQL procedure successfully completed.

SQL>
SQL> -- Modify the table to create skewed data on Country_id
SQL>
SQL> CREATE SEQUENCE ctest_pk_seq START WITH 200000
NOMAXVALUE;

Sequence created.

SQL>
SQL> INSERT INTO CTEST
  2  SELECT ctest_pk_seq.nextval,CUST_FIRST_NAME,
  3         CUST_LAST_NAME, CUST_GENDER,CUST_YEAR_OF_BIRTH,
  4         CUST_MARITAL_STATUS, CUST_STREET_ADDRESS,
  5         CUST_POSTAL_CODE,CUST_CITY, CUST_CITY_ID,
  6         CUST_STATE_PROVINCE,CUST_STATE_PROVINCE_ID,COUNTRY_ID,
  7         CUST_MAIN_PHONE_NUMBER, CUST_INCOME_LEVEL,
  8         CUST_CREDIT_LIMIT,CUST_EMAIL,CUST_TOTAL,CUST_TOTAL_ID,
  9         CUST_SRC_ID,CUST_EFF_FROM, CUST_EFF_TO,CUST_VALID
 10  FROM CTEST
 11  WHERE country_id = 52790;

18520 rows created.

SQL>
SQL> INSERT INTO CTEST
  2  SELECT ctest_pk_seq.nextval, CUST_FIRST_NAME,
  3         CUST_LAST_NAME, CUST_GENDER,CUST_YEAR_OF_BIRTH,
  4         CUST_MARITAL_STATUS, CUST_STREET_ADDRESS,
  5         CUST_POSTAL_CODE,CUST_CITY, CUST_CITY_ID,
  6         CUST_STATE_PROVINCE,CUST_STATE_PROVINCE_ID,COUNTRY_ID,
  7         CUST_MAIN_PHONE_NUMBER,CUST_INCOME_LEVEL,
  8         CUST_CREDIT_LIMIT,CUST_EMAIL,CUST_TOTAL,CUST_TOTAL_ID,
  9         CUST_SRC_ID,CUST_EFF_FROM, CUST_EFF_TO,CUST_VALID
 10  FROM CTEST
 11  WHERE country_id = 52790;

37040 rows created.

SQL>

```

2. Autotrace the SQL statement in the `bad_stats.sql` script and observe the execution plan and the differences between the estimated cost and the actual cost. The script is in the `$HOME/labs/sqltune` directory.

**Practice 10-2: Stale Statistics (continued)**

```

SQL> @bad_stats.sql
SQL>
SQL> -- Show the choice of FTS with old statistics
SQL> SET ECHO ON
SQL> SET AUTOTRACE TRACEONL
SQL>
SQL> SELECT CUST_ID, CUST_LAST_NAME, CUST_TOTAL
       2  FROM CTEST
       3  WHERE COUNTRY_ID = 52779;

3833 rows selected.

```

## Execution Plan

Plan hash value: 3591594933

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		2921	96393	406 (1)	00:00:05
* 1	TABLE ACCESS FULL	CTEST	2921	96393	406 (1)	00:00:05

## Predicate Information (identified by operation id):

1 - filter("COUNTRY\_ID"=52779)

## Statistics

```

2 recursive calls
1 db block gets
3162 consistent gets
2883 physical reads
176 redo size
86727 bytes sent via SQL*Net to client
3224 bytes received via SQL*Net from client
257 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)
3833 rows processed

```

```

SQL>
SQL> SET AUTOTRACE OFF
SQL> SET ECHO OFF

```

### Practice 10-2: Stale Statistics (continued)

3. What was the estimated number of rows?

---

 (2921)

What was the actual number of rows fetched?

\_\_\_\_\_ (3833)

What was the number of logical reads? \_\_\_\_\_  
(DB block gets + consistent gets)

4. Gather statistics against the table and indexes. Use the `GATHER_TABLE_STATS` procedure.

```
SQL> EXEC DBMS_STATS.GATHER_TABLE_STATS('SH','CTEST');
```

```
PL/SQL procedure successfully completed.
```

SQL&gt;

- Autotrace the SQL statement and observe the differences in the execution plan.

```
SQL> @bad_stats
```

SQL&gt;

```
SQL> -- Show the choice of FTS with old statistics
```

```
SQL> SET ECHO ON
```

```
SQL> SET AUTOTRACE TRACEONL
```

SQL&gt;

```
SQL> SELECT CUST_ID, CUST_LAST_NAME, CUST_TOTAL
```

2 FROM CTEST

```
3 WHERE COUNTRY_ID = 52779;
```

```
3833 rows selected.
```

## Execution Plan

Plan hash value: 4103564280

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	
-----							
0	SELECT STATEMENT		4247	136K	511 (0)	00:00:07	
1	TABLE ACCESS BY INDEX ROWID	CTEST	4247	136K	511 (0)	00:00:07	
* 2	INDEX RANGE SCAN	CTEST_COUNTRY	4247		12 (0)	00:00:01	

Predicate Information (identified by operation id):

```
2 - access( "COUNTRY_ID"=52779)
```

**Practice 10-2: Stale Statistics (continued)**

```

Statistics
-----
          0  recursive calls
          0  db block gets
       3132  consistent gets
       2874  physical reads
          0  redo size
      86727  bytes sent via SQL*Net to client
       3224  bytes received via SQL*Net from client
        257  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
       3833  rows processed

SQL>

```

6. What was the number of rows estimated? \_\_\_\_\_ (4247, may vary)  
 What was the number of rows fetched? \_\_\_\_\_ (3833)  
 What was the number of logical reads? \_\_\_\_\_ (varies)  
 The number of logical reads is a rough measure of the cost of the statement. There should be a small reduction in the number of logical reads between this execution and the previous execution of this statement.
7. Clean up the objects created for this scenario. Use the `cleanup_bad_stats.sql` script.

```

SQL> @cleanup_bad_stats.sql

SQL> DROP SEQUENCE ctest_pk_seq;

Sequence dropped.

SQL>
SQL> DROP TABLE CTEST PURGE;

Table dropped.

SQL>
SQL> EXIT
Disconnected from Oracle Database 11g Enterprise Edition
Release 11.1.0.6.0 - Production
With the Partitioning, Oracle Label Security, OLAP, Data
Mining
and Real Application Testing options
$

```

8. In this exercise, stale statistics cause the optimizer to choose a full-table scan (FTS) rather than an index access. This example shows a small cost reduction. The differences in cost can be magnified by repeated execution of the same statements and by larger differences in the statistics.

## Practices for Lesson 11

In this practice, you reduce the cost of queries by reducing the number of blocks that must be retrieved.



## Practice 11-1: Excess Blocks

In this practice, you see the differences in the explain plan and the costs of using a table that has more blocks than necessary. This condition can be caused in a number of ways: direct loads followed by deletes, inserts and deletes with a high PCTFREE setting, and batch deletes. The immediate solution is to shrink or reorganize the table; the long-term solution is to modify the processing or PCTUSED so that blocks are reused for inserts.

1. Create a table that uses more blocks than necessary. Use the `crctest.sql` script as the SH user. This script creates a table with the same data as the SH.CUSTOMERS table, but with a PCTFREE=80, so that it uses many more blocks than needed.

```
$ cd $HOME/labs/sqltune
$ sqlplus /nolog

SQL> connect sh/sh
Connected.

SQL> @crctest.sql
SQL>
SQL> -- create a new revision of the Customers table with a
SQL> -- large number of blocks.
SQL> set echo on
SQL>
SQL> DROP TABLE CTEST PURGE;
DROP TABLE CTEST PURGE
      *
ERROR at line 1:
ORA-00942: table or view does not exist

SQL>
SQL> CREATE TABLE CTEST
  2  PCTFREE 80 PCTUSED 10
  3  as SELECT * FROM customers;

Table created.

SQL>
SQL> EXEC DBMS_STATS.GATHER_TABLE_STATS('SH','CTEST');

PL/SQL procedure successfully completed.

SQL>
```

2. Run the ctest.sql script as the SH user with AUTOTRACE on.

```
SQL> @ctest.sql
SQL>
SQL> -- set autotrace on --
SQL> set echo on
SQL>
SQL> SET AUTOTRACE TRACEONLY
SQL>
SQL> select time_id, QUANTITY_SOLD, AMOUNT_SOLD
       2   from sales s, ctest c
       3   where c.cust_id = s.cust_id and CUST_FIRST_NAME='Dina'
       4   order by time_id;
```

895 rows selected.

#### Execution Plan

Plan hash value: 2054825740

Id	Operation	Name	Rows	Bytes	TempSpc	Cost (%CPU)	Time	Pstart	Pstop
0	SELECT STATEMENT		5557	179K		2453 (1)	00:00:30		
1	SORT ORDER BY		5557	179K	272K	2453 (1)	00:00:30		
* 2	HASH JOIN		5557	179K		2402 (1)	00:00:29		
* 3	TABLE ACCESS FULL	CTEST	43	516		1912 (1)	00:00:23		
4	PARTITION RANGE ALL		918K	18M		498 (4)	00:00:06	1	28
5	TABLE ACCESS FULL	SALES	918K	18M		498 (4)	00:00:06	1	28

#### Predicate Information (identified by operation id):

```
2 - access("C"."CUST_ID"="S"."CUST_ID")
3 - filter("CUST_FIRST_NAME"='Dina')
```

#### Statistics

```
13516 recursive calls
      0 db block gets
11455 consistent gets
 8658 physical reads
 1242 redo size
```

```

      23039 bytes sent via SQL*Net to client
      1068 bytes received via SQL*Net from client
        61 SQL*Net roundtrips to/from client
      111 sorts (memory)
        0 sorts (disk)
      895 rows processed

SQL>
SQL> SET AUTOTRACE OFF
SQL>

```

- Note the cost, both in terms of the number of physical reads and logical reads (consistent gets + DB block gets), and the estimated cost.  
 Physical reads \_\_\_\_\_ (8658, may vary)  
 Logical reads \_\_\_\_\_ (11455, may vary)  
 Estimated cost \_\_\_\_\_ (2453)
- Reorganize the table to use fewer blocks. Run the `reorg_ctest.sql` script. This script sets `PCTFREE` to 10, and then moves the table. This script also checks the number of blocks used before and after the reorganization.

```

SQL> @reorg_ctest.sql
SQL> -- Rebuild the table with higher density
SQL> SET ECHO ON
SQL>
SQL> SELECT BLOCKS, EMPTY_BLOCKS FROM USER_TABLES
       2 WHERE TABLE_NAME = 'CTEST';

      BLOCKS EMPTY_BLOCKS
-----
      7038             0

SQL>
SQL> ALTER TABLE CTEST
       2 PCTFREE 10;

Table altered.

SQL>
SQL> ALTER TABLE CTEST MOVE;

Table altered.

SQL>
SQL> EXEC DBMS_STATS.GATHER_TABLE_STATS('SH','CTEST');
PL/SQL procedure successfully completed.

SQL>
SQL> SELECT BLOCKS, EMPTY_BLOCKS FROM USER_TABLES
       2 WHERE TABLE_NAME = 'CTEST';

      BLOCKS EMPTY_BLOCKS
-----

```

1485	0
------	---

SQL>

5. Run the ctest.sql script as the SH user with AUTOTRACE on.

```
SQL> @ctest.sql
SQL>
SQL> -- set autotrace on --
SQL> set echo on
SQL>
SQL> SET AUTOTRACE TRACEONLY
SQL>
SQL> select time_id, QUANTITY_SOLD, AMOUNT_SOLD
       2   from sales s, ctest c
       3   where c.cust_id = s.cust_id and CUST_FIRST_NAME='Dina'
       4   order by time_id;

895 rows selected.

Execution Plan
-----
Plan hash value: 2054825740
```

Id	Operation	Name	Rows	Bytes	TempSpc	Cost (%CPU)	Time	Pstart	Pstop
0	SELECT STATEMENT		5557	179K		948 (3)	00:00:12		
1	SORT ORDER BY		5557	179K	536K	948 (3)	00:00:12		
* 2	HASH JOIN		5557	179K		897 (3)	00:00:11		
* 3	TABLE ACCESS FULL	CTEST	43	516		405 (1)	00:00:05		
4	PARTITION RANGE ALL		918K	18M		489 (4)	00:00:06	1	28
5	TABLE ACCESS FULL	SALES	918K	18M		498 (4)	00:00:06	1	28

Predicate Information (identified by operation id):

```
-----
2 - access("C"."CUST_ID"="S"."CUST_ID")
3 - filter("CUST_FIRST_NAME"='Dina')
```

Statistics

```
-----
15301 recursive calls
0 db block gets
6306 consistent gets
```

```

      3217  physical reads
        0  redo size
    23039  bytes sent via SQL*Net to client
    1068  bytes received via SQL*Net from client
       61  SQL*Net roundtrips to/from client
      147  sorts (memory)
        0  sorts (disk)
      895  rows processed

SQL>
SQL> SET AUTOTRACE OFF

```

6. Note the various costs. Write down the number of physical reads and logical reads (consistent gets + DB gets), and the estimated cost.

Physical reads \_\_\_\_\_ (3217, may vary)

Logical reads \_\_\_\_\_ ( 6306, may vary)

Estimated cost \_\_\_\_\_ (948)

The actual numbers that you see may vary from those in the example, but you should see a reduction in the number of logical and physical reads and the estimated cost. These reductions are due to reducing the number of physical blocks used to hold the table rows. This effect is most pronounced with full-table scans. But even with index access methods, if the table uses more blocks, it will require more memory buffers and more block visits to access the same number of rows.

7. Clean up the objects for this exercise.

```

SQL> DROP TABLE CTEST PURGE;
SQL> EXIT

```

## Practice 11-2: Coalescing an Index

In this practice, you observe the effect of coalescing an index on a SQL statement and an execution plan.

1. The `coalesce_setup.sql` script creates the TESTLM tablespace, the CTEST table, and the CTEST\_PK index, with excess space in the index. CTEST is created as an empty version of the SH.CUSTOMERS table, and then populated with four times the rows from CUSTOMERS. To make the index sparse, every second row is deleted based on CUST\_ID, which is unique and is generated by a sequence.

```
$ cd $HOME/labs/sqltune
$ sqlplus /nolog

SQL> SET ECHO ON

SQL> @coalesce_setup.sql

SQL> CONNECT / as sysdba
Connected.
SQL>
SQL> DROP TABLESPACE TESTLM INCLUDING CONTENTS AND DATAFILES;
DROP TABLESPACE TESTLM INCLUDING CONTENTS AND DATAFILES
*
ERROR at line 1:
ORA-00959: tablespace 'TESTLM' does not exist

SQL>
SQL> CREATE TABLESPACE TESTLM
  2  DATAFILE '/u01/app/oracle/oradata/orcl/testlm1.dbf'
  3  SIZE 100M
  4  AUTOEXTEND ON NEXT 10M MAXSIZE 300M
  5  SEGMENT SPACE MANAGEMENT MANUAL;

Tablespace created.

SQL>
SQL> CONNECT sh/sh
Connected.

SQL>
SQL> -- Create a table with indexes that have large number of
SQL> -- partially empty blocks and will benefit from an index
SQL> -- coalesce
SQL>
SQL> -- Create a table, indexes and gather statistics
SQL> DROP SEQUENCE ctest_pk_seq;
DROP SEQUENCE ctest_pk_seq
*
ERROR at line 1:
```

## Practice 11-2: Coalescing an Index (continued)

ORA-02289: sequence does not exist

```
SQL> DROP TABLE CTEST PURGE;
DROP TABLE CTEST PURGE
      *
```

ERROR at line 1:

ORA-00942: table or view does not exist

SQL>

```
SQL> CREATE TABLE CTEST TABLESPACE TESTLM
      2 AS SELECT *
      3 FROM CUSTOMERS where 1=2;
```

Table created.

SQL>

```
SQL> ALTER TABLE CTEST ADD Constraint CTEST_PK
      2 Primary KEY (cust_id);
```

Table altered.

```
SQL> ALTER TABLE CTEST ADD (DUMMY VARCHAR2(300));
```

Table altered.

SQL>

SQL>

```
SQL> CREATE SEQUENCE ctest_pk_seq START WITH 200000
NOMAXVALUE;
```

Sequence created.

SQL>

```
SQL> INSERT INTO CTEST
      2 SELECT ctest_pk_seq.nextval, CUST_FIRST_NAME,
      3 CUST_LAST_NAME, CUST_GENDER, CUST_YEAR_OF_BIRTH,
      4 CUST_MARITAL_STATUS, CUST_STREET_ADDRESS, CUST_POSTAL_CODE,
      5 CUST_CITY, CUST_CITY_ID, CUST_STATE_PROVINCE,
      6 CUST_STATE_PROVINCE_ID, COUNTRY_ID, CUST_MAIN_PHONE_NUMBER,
      7 CUST_INCOME_LEVEL, CUST_CREDIT_LIMIT, CUST_EMAIL,
      8 CUST_TOTAL, CUST_TOTAL_ID, CUST_SRC_ID, CUST_EFF_FROM,
      9 CUST_EFF_TO, CUST_VALID,
     10 CUST_FIRST_NAME || ' ' || CUST_LAST_NAME
     11 FROM CUSTOMERS;
```

55500 rows created.

SQL>

SQL>

```
SQL> INSERT INTO CTEST
```

**Practice 11-2: Coalescing an Index (continued)**

```

2  SELECT ctest_pk_seq.nextval, CUST_FIRST_NAME,
3         CUST_LAST_NAME,CUST_GENDER, CUST_YEAR_OF_BIRTH,
4  CUST_MARITAL_STATUS,CUST_STREET_ADDRESS,CUST_POSTAL_CODE,
5         CUST_CITY,CUST_CITY_ID,CUST_STATE_PROVINCE,
6  CUST_STATE_PROVINCE_ID,COUNTRY_ID,CUST_MAIN_PHONE_NUMBER,
7         CUST_INCOME_LEVEL,CUST_CREDIT_LIMIT,CUST_EMAIL,
8         CUST_TOTAL,CUST_TOTAL_ID,CUST_SRC_ID,CUST_EFF_FROM,
9         CUST_EFF_TO,CUST_VALID,
10        CUST_FIRST_NAME||' '||CUST_LAST_NAME
11  FROM CUSTOMERS;

```

55500 rows created.

SQL>

SQL>

SQL> INSERT INTO CTEST

```

2  SELECT ctest_pk_seq.nextval, CUST_FIRST_NAME,
3         CUST_LAST_NAME,CUST_GENDER, CUST_YEAR_OF_BIRTH,
4  CUST_MARITAL_STATUS,CUST_STREET_ADDRESS,CUST_POSTAL_CODE,
5         CUST_CITY,CUST_CITY_ID,CUST_STATE_PROVINCE,
6  CUST_STATE_PROVINCE_ID,COUNTRY_ID,CUST_MAIN_PHONE_NUMBER,
7         CUST_INCOME_LEVEL,CUST_CREDIT_LIMIT,CUST_EMAIL,
8         CUST_TOTAL,CUST_TOTAL_ID,CUST_SRC_ID,CUST_EFF_FROM,
9         CUST_EFF_TO,CUST_VALID,
10        CUST_FIRST_NAME||' '||CUST_LAST_NAME
11  FROM CUSTOMERS;

```

55500 rows created.

SQL>

SQL>

SQL> INSERT INTO CTEST

```

2  SELECT ctest_pk_seq.nextval, CUST_FIRST_NAME,
3         CUST_LAST_NAME,CUST_GENDER, CUST_YEAR_OF_BIRTH,
4  CUST_MARITAL_STATUS,CUST_STREET_ADDRESS,CUST_POSTAL_CODE,
5         CUST_CITY,CUST_CITY_ID,CUST_STATE_PROVINCE,
6  CUST_STATE_PROVINCE_ID,COUNTRY_ID,CUST_MAIN_PHONE_NUMBER,
7         CUST_INCOME_LEVEL,CUST_CREDIT_LIMIT,CUST_EMAIL,
8         CUST_TOTAL,CUST_TOTAL_ID,CUST_SRC_ID,CUST_EFF_FROM,
9         CUST_EFF_TO,CUST_VALID,
10        CUST_FIRST_NAME||' '||CUST_LAST_NAME
11  FROM CUSTOMERS;

```

55500 rows created.

SQL>

SQL> DELETE FROM CTEST

```

2  WHERE mod(cust_id,2) = 1;

```

111000 rows deleted.

SQL>



**Practice 11-2: Coalescing an Index (continued)**

```
SQL> commit;

Commit complete.

SQL>
SQL> EXEC DBMS_STATS.GATHER_TABLE_STATS('SH','CTEST',-
>    method_opt=>'FOR ALL INDEXED COLUMNS SIZE AUTO');

PL/SQL procedure successfully completed.

SQL>
```

2. Execute the SQL statement in the `coalesce_idx.sql` script. The script sets AUTOTRACE TRACEONLY.

What is the access method? \_\_\_\_\_ (Full-table scan)

What is the estimated cost? \_\_\_\_\_ (1693)

What is the number of consistent reads? \_\_\_\_\_ (8091, may vary)

```
SQL> @coalesce_idx_1.sql
SQL> -- Will yield different plans before and after index
coalesce
SQL>
SQL> Set AUTOTRACE TRACEONLY
SQL> select /*BEFORE*/ cust_id,cust_last_name, cust_first_name
      2   from ctest
      3  where cust_id < 257500
      /

28749 rows selected.
```

## Execution Plan

Plan hash value: 3591594933

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		28749	1094K	1691 (1)	00:00:21
* 1	TABLE ACCESS FULL	CTEST	28749	1094K	1691 (1)	00:00:21

Predicate Information (identified by operation id):

1 - filter("CUST\_ID"<257500)

## Statistics

```
1 recursive calls
0 db block gets
8100 consistent gets
```

**Practice 11-2: Coalescing an Index (continued)**

```

        6228  physical reads
          0   redo size
    613586  bytes sent via SQL*Net to client
    21495   bytes received via SQL*Net from client
     1918   SQL*Net roundtrips to/from client
          0   sorts (memory)
          0   sorts (disk)
    28749   rows processed

SQL>

```

3. Execute the `coalesce_idx_size.sql` script to view index statistics.

```

SQL>
SQL> @coalesce_idx_size
SQL> -- Find the important index stats before and
SQL> -- after coalesce
SQL>
SQL> SELECT Blevel, LEAF_BLOCKS, DISTINCT_KEYS,
CLUSTERING_FACTOR
   2  FROM USER_INDEXES
   3  WHERE index_name = 'CTEST_PK';

      BLEVEL LEAF_BLOCKS DISTINCT_KEYS CLUSTERING_FACTOR
-----
          1          417         111000              6224

SQL>

```

4. Coalesce the CTEST\_PK index.

```

SQL> @coalesce_fix
SQL> -- Fix the index (Coalesce)
SQL>
SQL> ALTER INDEX CTEST_PK COALESCE;

Index altered.

SQL>
SQL> -- GATHER STATS
SQL>
SQL> EXEC DBMS_STATS.GATHER_TABLE_STATS('SH','CTEST',-
   2  METHOD_OPT=>'FOR ALL INDEXED COLUMNS SIZE AUTO');

PL/SQL procedure successfully completed.

SQL>

```

## Practice 11-2: Coalescing an Index (continued)

5. View index statistics again. Notice the difference in the number of leaf blocks.

```
SQL> @coalesce_idx_size
SQL> -- Find the important index stats before and
SQL> -- after coalesce
SQL>
SQL> SELECT Blevel, LEAF_BLOCKS, DISTINCT_KEYS,
CLUSTERING_FACTOR
2 FROM USER_INDEXES
3 WHERE index_name = 'CTEST_PK';

      BLEVEL LEAF_BLOCKS DISTINCT_KEYS CLUSTERING_FACTOR
-----
1          232          111000          6224

SQL>
```

6. Execute the SQL statement in the coalesce\_idx.sql script again.

```
SQL> @coalesce_idx_2.sql
SQL> -- Will yield different plans before and after index
coalesce
SQL>
SQL> Set AUTOTRACE TRACEONLY
SQL> select /*AFTER*/ cust_id, cust_last_name, cust_first_name
2 from ctest
3 where cust_id < 257500
4 /

28749 rows selected.
```

### Execution Plan

Plan hash value: 3972627342

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		28749	1094K	1676 (1)	00:00:21
1	TABLE ACCESS BY INDEX ROWID	CTEST	28749	1094K	1676 (1)	00:00:21
* 2	INDEX RANGE SCAN	CTEST_PK	28749		62 (0)	00:00:01

Predicate Information (identified by operation id):

2 - access("CUST\_ID"<257500)

## Practice 11-2: Coalescing an Index (continued)

Statistics

```
-----
      1 recursive calls
      0 db block gets
    5393 consistent gets
    1648 physical reads
      0 redo size
  613486 bytes sent via SQL*Net to client
   21495 bytes received via SQL*Net from client
    1918 SQL*Net roundtrips to/from client
      0 sorts (memory)
      0 sorts (disk)
   28749 rows processed
```

SQL>

SQL> SET AUTOTRACE OFF

SQL> **exit**

7. Observe the differences in the estimated cost and the number of logical reads (consistent gets).  
What is the access method? \_\_\_\_\_ (index-range scan)  
What is the estimated cost? \_\_\_\_\_ (1676)  
How many consistent reads were performed? \_\_\_\_\_ (5293, may vary)
8. Clean up this practice—execute the `./cleanup 10 4` script in the `$HOME/workshops` directory.

```
$ cd $HOME/workshops
```

```
$ ./cleanup OPT
```

```
Tablespace dropped.
```

```
$
```

## Practices for Lesson 12

The goal of this practice is to use SQL Performance Analyzer to identify the SQL statements that regress when the environment is upgraded.

## Practice 12-1: Using the SQL Performance Analyzer

In this practice, you simulate exporting a SQL Tuning Set (STS) from a 10g database and import it back into an 11g test environment. There, you access the performance of the SQL statements that you imported before upgrading the 10g database.

1. From a terminal window, referred to as the “first session,” execute the `setup_SPAbig10g.sh` script to set up your simulated 10g environment. In this simulated environment, the `OPTIMIZER_FEATURES_ENABLED` parameter is set to 10.2.0.2.

```
$ cd $HOME/labs/SPA
$ cat setup_SPAbig10g.sh

#!/bin/bash

rm /u01/app/oracle/admin/orcl/dpdump/apps.dmp
rm /u01/app/oracle/admin/orcl/dpdump/appsandstage.dmp

cp /home/oracle/labs/SPA/apps.dmp
/u01/app/oracle/admin/orcl/dpdump/apps.dmp

sqlplus /nolog <<FIN!
connect / as sysdba

@$HOME/workshops/spwkshSPA.sql

shutdown immediate

startup

set echo on

exec dbms_sqltune.drop_sqlset('STS_JFV','SYS');

drop user apps cascade;

host impdp system/oracle directory=DATA_PUMP_DIR
dumpfile=apps.dmp

select distinct last_analyzed from dba_tab_statistics where
owner='APPS';

EXECUTE DBMS_STATS.LOCK_SCHEMA_STATS('APPS');

ALTER DATABASE TEMPFILE
'/u01/app/oracle/oradata/orcl/temp01.dbf'
AUTOEXTEND ON NEXT 6400K MAXSIZE UNLIMITED;

FIN!
```

**Practice 12-1: Using the SQL Performance Analyzer (continued)**

```
$ ./setup_SPAbig10g.sh
```

2. **(Perform steps 2 and 3 at the same time.)** Generate a SQL Tuning Set (STS) called STS\_JFV that captures SQL statements from the cursor cache for approximately 12 minutes every five seconds. Make sure that you try to capture only those statements that come out of the SQL\_JFV module in the APPS schema. Also, this STS should belong to the SYS user. Use the capsts10g.sh script to perform this step.

```
$ cd /home/oracle/labs/SPA
$ cat capsts10g.sh
#!/bin/bash

sqlplus / as sysdba <<FIN!

SET ECHO ON;
SET TIMING ON;

begin
DBMS_SQLTUNE.CREATE_SQLSET (sqlset_name => 'STS_JFV');

dbms_sqltune.capture_cursor_cache_sqlset(
sqlset_name => 'STS_JFV' ,
basic_filter=> q'# module
like 'DWH_TEST%' and sql_text not like '%applicat%' and
parsing_schema_name in ('APPS') #' ,
time_limit => 8*60,
repeat_interval => 5);
end ;
/
show errors

FIN!

$ ./capsts10g.sh
```

3. **(Perform steps 2 and 3 at the same time.)** From a second terminal window, connected as the oracle user, execute your workload by using the wrkl10g\_jfv.sh script. This script runs a workload of 45 statements that are going to be captured in STS\_JFV automatically.

```
-- Second session
$ cd /home/oracle/labs/SPA
$ ./wrkl10g_jfv.sh

...
SQL> @6 Results in 1296 Ticks
SQL> SQL> SQL> SQL> @Statement 45
SQL> SQL> SQL> 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20 21 22 23
```

**Practice 12-1: Using the SQL Performance Analyzer (continued)**

```

24    25    26    27    28    29    30    31    32    33    34    35    36
37    38    39    40    41    42    43    44    45    46    47    48    49
50    51    52    53    54    55    56    57    58    59    60    61    62
63    64    65    66    67    68    69    70    71    72    73    74    75
76    77    78    79    80    81    82    83    84    85    86    87    88
89    90    91    92    93    94    95    96    97    98    99   100   101
102   103   104
10 rows selected.

Elapsed: 00:00:30.24

Statistics
-----
          92 recursive calls
           0 db block gets
6871559 consistent gets
10770 physical reads
    116 redo size
    2003 bytes sent via SQL*Net to client
    3147 bytes received via SQL*Net from client
         2 SQL*Net roundtrips to/from client
         2 sorts (memory)
         0 sorts (disk)
        10 rows processed

SQL> @10 Results in 1188 Ticks
SQL> SQL> SQL> SQL>
PL/SQL procedure successfully completed.

SQL> SQL> we are done
SQL> SQL> SQL> SQL> @End 2010:FEB-11:18:50:40

SQL> SQL> Disconnected ...

```

- After approximately eight minutes, both sessions should have finished. Check the content of STS\_JFV, and stage it in a table called APPS.STS\_JFV\_TAB. Use the stage\_sts.sh script to perform this step.

```

$ cd /home/oracle/labs/SPA
$ cat stage_sts.sh

#!/bin/bash
sqlplus /nolog <<FIN!

connect / as sysdba

set echo on
select name,statement_count from dba_sqlset;

drop table apps.sts_jfv_tab purge;

exec DBMS_SQLTUNE.CREATE_STGTAB_SQLSET('STS_JFV_TAB','APPS');

```



**Practice 12-1: Using the SQL Performance Analyzer (continued)**

```

exec
DBMS_SQLTUNE.PACK_STGTAB_SQLSET('STS_JFV','SYS','STS_JFV_TAB',
'APPS');
FIN!

$ ./stage_sts.sh

```

- Using Data Pump Export, export the APPS schema to the default Data Pump 10g directory. Use the `exportapps.sh` script to perform this step.

```

$ cd /home/oracle/labs/SPA
$ cat exportapps.sh

#!/bin/bash

sqlplus /nolog <<FIN!
connect apps/apps

drop table plan_table purge;

FIN!

rm /u01/app/oracle/admin/orcl/dpdump/appsandstage.dmp

expdp system/oracle_4U directory=DATA_PUMP_DIR
dumpfile=appsandstage schemas=apps

$ ./exportapps.sh

```

- In real 10g and 11g environments, you would copy the generated dump file from the default Data Pump 10g directory to the default Data Pump 11g directory. But in this simulated environment, they are the same directory. So nothing needs to be done.
- Now, restart your 11g environment to restore the database to an 11g environment. Use the `setup_SPAbig11g.sh` script to perform this step.

```

$ ./setup_SPAbig11g.sh

SQL> SQL> Connected.
SQL> SQL> Database closed.
Database dismounted.
ORACLE instance shut down.
SQL>
SQL> ORACLE instance started.

Total System Global Area  836976640 bytes
Fixed Size                  1339740 bytes

```

## Practice 12-1: Using the SQL Performance Analyzer (continued)

```
Variable Size          796921508 bytes
Database Buffers       25165824 bytes
Redo Buffers           13549568 bytes
Database mounted.
Database opened.
SQL> SQL> Disconnected ...
```

8. Using Data Pump Import, import the APPS schema into your 11g system. Use the `importapps.sh` script to perform this step.

```
$ cd /home/oracle/labs/SPA
$ cat ./importapps.sh

#!/bin/bash

sqlplus /nolog <<FIN!

Connect / as sysdba
drop user apps cascade;

host impdp system/oracle_4U directory=DATA_PUMP_DIR
dumpfile=appsandstage

FIN!

$ ./importapps.sh
```

9. Unpack the previously imported staging table in the SYS schema. Use the `unpack_sts.sh` script to perform this step.

```
$ cd /home/oracle/labs/SPA
$ cat unpack_sts.sh

#!/bin/bash

sqlplus /nolog <<FIN!

Connect / as sysdba
set echo on

exec
DBMS_SQLTUNE.UNPACK_STGTAB_SQLSET('STS_JFV','SYS',TRUE,'STS_JF
V_TAB','APPS');

alter system flush buffer_cache;
alter system flush shared_pool;

FIN!

$ ./unpack_sts.sh
```

## Practice 12-1: Using the SQL Performance Analyzer (continued)

10. Use Enterprise Manager Database Control, as the SYS user, to test the behavior of STS\_JFV in the simulated 10g environment and compare it to the 11g environment. You will do this by changing the OPTIMIZER\_FEATURES\_ENABLED parameter. What are your conclusions?

**Note:** Screenshots of some of the steps described in the table are provided, as noted in the Page column, below the table. Refer to these screenshots as necessary.

Step	Page	Action		
a	Database home	Click the Software and Support tab.		
b	Software and Support page	Click SQL Performance Analyzer in the Real Application Testing section.		
c	SQL Performance Analyzer page	Click Parameter Change.		
d	Parameter Change (See the example screenshot, which follows the table.)	Set values.	Field	Value
			Task Name	SPA_JFV1
			SQL Tuning Set	STS_JFV
			Creation Method	Execute SQLs
			Per-SQL Time Limit	Unlimited
			Parameter Name	optimizer_features_enabled (click the select, “flashlight”, icon)
			Base Value	10.2.0.1
			Changed Value	11.2.0.1
			Comparis	Elapsed Time

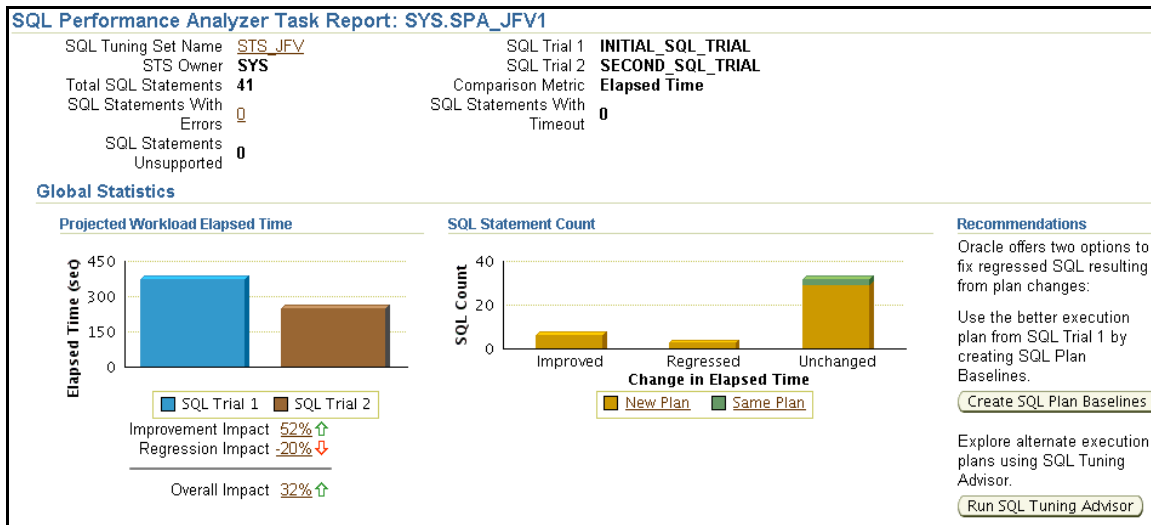
## Practice 12-1: Using the SQL Performance Analyzer (continued)

			on Metric	
			Schedule	Immediately
		Click Submit.		
e	SQL Performance Analyzer	Let the screen refresh until the Status field shows Completed for your submitted task (SPA_JFV1). It may take up to 22 minutes to execute.		
		Click the SPA_JFV1 link.		
f	SQL Performance Analyzer Task: SYS.SPA_JFV1	This page shows the SQL Trials and the SQL Trial Comparisons. Click the eyeglass icon in the SQL Trial Comparisons table in the Comparison Report column. There should be only one entry in this table.		
g	SQL Performance Analyzer Task Report: SYS.SPA_JFV1  (See the example screenshots, which follow the table.)	You can tell that the second trial was faster than the first one. The second trial corresponds to the run that used the 11g optimizer features. The benefit depends on the statements that were captured and executed. You will see the number of the statements that use a new plan with 11g optimization. However, some statements regressed with 11g optimization.		
		From “Top 10 SQL Statements Based on Impact on Workload,” make a note of the SQL ID corresponding to the statements that have the highest negative impact on your workload. You use this information in Practice 13-1.		
		Click the SQL ID link corresponding to the most regressing one.		
h	SQL Details	You can see the text of your statement and the different execution plans that were used during both trials.		
		Click the Back button on the browser.		
i	SQL Performance Analyzer Task Result	Click Run SQL Tuning Advisor in the Recommendations section.		
j	Schedule SQL Tuning Task	Set values.	Field	Value
			Tuning Task Name	TUNE_JFV1
			Schedule	Immediately

## Practice 12-1: Using the SQL Performance Analyzer (continued)

		Click OK.
k	SQL Performance Analyzer Task Result	Click the SQL Tune Report SYS.TUNE_JFV1 link in the Recommendations section.
		Use the browser refresh button until you see a report that includes graphics. This may take a minute or longer.
l	SQL Tuning Result Summary:SYS.TUNE_JFV1 (see screenshot)	<p>On this page, you can see that there are SQL Statements that can be improved by using profiles, indexes, or alternative SQL plans (using SQL Baselines). Using one of these recommendations, you can reduce or eliminate the regressions.</p> <p>Do not implement at this time; the following lessons explore these options in more detail.</p>

g) The following screenshot displays the SQL Performance Analyzer Task Result: SYS.SPA\_JFV1 page:

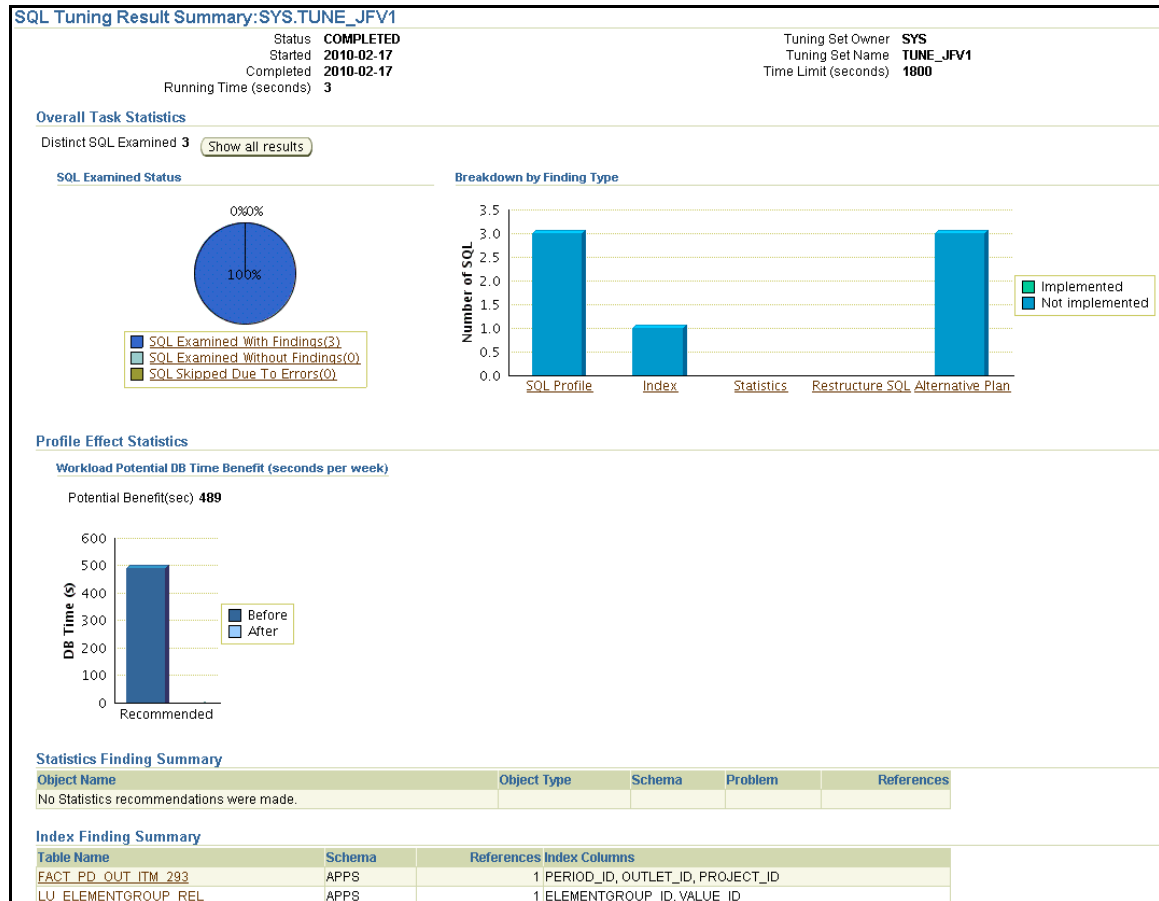


## Practice 12-1: Using the SQL Performance Analyzer (continued)

Top 10 SQL Statements Based on Impact on Workload						
	SQL ID	Net Impact on Workload (%)	Elapsed Time (sec)		Net Impact on SQL (%)	New Plan
			SQL Trial 1	SQL Trial 2		
↓	<a href="#">gkc6w4zhaysbk</a>	-18.000	103.999	171.012	-64.440	Y
↑	<a href="#">gfw9mbv2h44ns</a>	9.290	34.936	0.368	98.950	Y
↑	<a href="#">21t61c8b39njg</a>	9.200	34.574	0.326	99.060	Y
↑	<a href="#">94jmd58x6ch6d</a>	9.190	34.611	0.407	98.820	Y
↑	<a href="#">dx1c9zbr6w8h6</a>	8.870	33.329	0.321	99.040	Y
↑	<a href="#">2pg3srqh3qasz</a>	8.030	30.244	0.351	98.840	Y
↑	<a href="#">1wfywkvvcwm2sp</a>	7.410	73.676	46.098	37.430	Y
↓	<a href="#">7866641pah2zg</a>	-1.010	3.457	7.235	-109.300	Y
↓	<a href="#">cpm5u0m6b68k5</a>	-1.010	3.543	7.317	-106.560	Y
↑	<a href="#">dvaxhpys7hpdv</a>	0.550	2.923	0.860	70.560	Y

## Practice 12-1: Using the SQL Performance Analyzer (continued)

1) The following screenshot displays the SQL Tuning Result Summary:SYS.TUNE\_JFV1.



## Practices for Lesson 13

SQL Plan Management (SPM) is an Oracle Database 11g feature that provides controlled execution plan evolution.

With SPM, the optimizer automatically manages execution plans and ensures that only known or verified plans are used.

When a new plan is found for a SQL statement, it will not be used until it has been verified to have comparable or better performance than the current plan.

SPM has three main components:

1. Plan Capture
2. Plan Selection
3. Plan Verification

In these practices, you see each of these components in action.



## Practice 13-1: Seeding SQL Plan Baselines from SQL Performance Advisor

In this practice, you use the result of your SQL Performance Analyzer comparison trial to seed SQL plan baselines for regressing SQL statements to avoid future plan regressions. The regressions could have come from a variety of sources. Regressions could be due to changes in statistics or changes in database parameters.

1. Use Enterprise Manager to check whether there are baselines for your most regressing statements. Then use Enterprise Manager to seed SQL Plan Baselines for your most regressing statements.

**Note:** Screenshots of some of the steps described in the table are provided, as noted in the Page column, below the table. Refer to these screenshots as necessary.

Step	Page	Action
a	Current EM page	Click the Database tab.
b	Database home	Click the Server tab.
c	Server	Click SQL Plan Control in the Query Optimizer section.
d	SQL Plan Control	Click the SQL Plan Baseline tab.
		You should not see any baseline for your regressing statements.
		Click the Database tab.
e	Database home	Click the Software and Support tab.
f	Software and Support	Click the SQL Performance Analyzer link in the Real Application Testing section.
g	SQL Performance Analyzer	Click the SPA_JFV1 link in the SQL Performance Analyzer Tasks table.
h	SQL Performance Analyzer Task: SYS.SPA_JFV1 (See the example screenshot, which follows the table.)	Click the eyeglass icon in the Comparison Report column for your first replay trial comparison.  Or you can click the View Latest Report link at the top of the page to view the same report, when there is only one Comparison Report.
i	SQL Performance Analyzer Task Result:	Click Create SQL Plan Baselines in the Recommendations section.

**Practice 13-1: Seeding SQL Plan Baselines from SQL Performance Advisor (continued)**

	SYS.SPA_JFV1	
j	SQL Performance Analyzer Task Result: SYS.SPA_JFV1  (See the example screenshot, which follows the table.)	You should now see all regressing statements that meet the criteria. If the impact is before a certain threshold, they will not be listed.
		Set values
		Field
		Value
		Job Name
		baseline_jfv1
		Schedule
		Immediately
		Click OK.
k	SQL Performance Analyzer Task Result: SYS.SPA_JFV1	You should see an Information message, at the top of the page, saying that your job has been successfully submitted.
		Click the Database tab.
l	Database home	Click the Server tab.
m	Server	Click SQL Plan Control.
n	SQL Plan Control	Click the SQL Plan Baseline tab.
		You should now see the SQL plan baselines that were created. You should have one baseline for each regressing statement.

## Practice 13-1: Seeding SQL Plan Baselines from SQL Performance Advisor (continued)

h) The following screenshot displays the SQL Performance Analyzer Task: SYS.SPA\_JFV1 page:

**SQL Performance Analyzer Task: SYS.SPA\_JFV1**

[View Latest Report](#) Page Refreshed Feb 17, 2010 7:05:54 AM UTC [Refresh](#)

The SQL Performance Analyzer Task is a container for experimental results of executing a specific SQL Tuning Set under changed environmental conditions and assessing the impact of environmental changes on STS execution performance.

[▶ SQL Tuning Set](#)

[▼ SQL Trials](#)

A SQL Trial captures the execution performance of the SQL Tuning Set under specific environmental conditions. [Create SQL Trial](#)

SQL Trial Name	Description	Created	SQL Executed	Status
INITIAL_SQL_TRIAL	parameter optimizer_features_enable set to '10.2.0.1'	2/17/10 6:07 AM	Yes	COMPLETED
SECOND_SQL_TRIAL	parameter optimizer_features_enable set to '11.2.0.1'	2/17/10 6:17 AM	Yes	COMPLETED

[▼ SQL Trial Comparisons](#)

Compare SQL Trials to assess change impact of environmental differences on SQL Tuning Set execution costs. [Run SQL Trial Comparison](#)

Trial 1 Name	Trial 2 Name	Compare Metric	Created	Status	Comparison Report	SQL Tune Report
INITIAL_SQL_TRIAL	SECOND_SQL_TRIAL	Elapsed Time	2/17/10 6:26 AM	COMPLETED	<a href="#">Report</a>	<a href="#">Report</a>

i) The following screenshot displays the SQL Performance Analyzer Task Report: SYS.SPA\_JFV1 page:

**SQL Performance Analyzer Task Report: SYS.SPA\_JFV1**

SQL Tuning Set Name: [SYS\\_JFV](#) SQL Trial 1: [INITIAL\\_SQL\\_TRIAL](#)  
 STS Owner: **SYS** SQL Trial 2: [SECOND\\_SQL\\_TRIAL](#)  
 Total SQL Statements: **40** Comparison Metric: **Elapsed Time**  
 SQL Statements With Errors: [0](#) SQL Statements With Timeout: **0**  
 SQL Statements Unsupported: **0**

**Global Statistics**

**Projected Workload Elapsed Time**

■ SQL Trial 1 ■ SQL Trial 2

Improvement Impact: **56%** ↑  
 Regression Impact: **-25%** ↓  
 Overall Impact: **31%** ↑

**SQL Statement Count**

■ New Plan ■ Same Plan

**Recommendations**

Oracle offers two options to fix regressed SQL resulting from plan changes:

Use the better execution plan from SQL Trial 1 by creating SQL Plan Baselines.

[Create SQL Plan Baselines](#)

SQL Tune Report: [SYS.TUNE\\_JFV1](#)

**Top 10 SQL Statements Based on Impact on Workload**

SQL ID	Net Impact on Workload (%)	Elapsed Time (sec)		Net Impact on SQL (%)	New Plan
		SQL Trial 1	SQL Trial 2		
<a href="#">gkc6w4zhaysbk</a>	-21.950	99.152	165.800	-67.220	Y
<a href="#">21t61c8b39njq</a>	12.560	38.582	0.456	98.820	Y
<a href="#">dx1c9zbr6w8h6</a>	11.520	35.386	0.405	98.860	Y
<a href="#">94imd58x6ch6d</a>	11.380	35.082	0.522	98.510	Y
<a href="#">qfw9mbv2h44ns</a>	11.010	33.881	0.458	98.650	Y
<a href="#">2pq3srqh3qasz</a>	9.710	29.902	0.416	98.610	Y
<a href="#">cpm5u0m6b68k5</a>	-1.570	3.721	8.499	-128.430	Y
<a href="#">7866641pah2zq</a>	-1.480	4.091	8.582	-109.750	Y
<a href="#">dvaxhpvs7hpdz</a>	0.850	3.173	0.585	81.550	Y
<a href="#">av7s4k9vz34hk</a>	0.520	1.797	0.233	87.050	Y

## Practice 13-1: Seeding SQL Plan Baselines from SQL Performance Advisor (continued)

j) The following screenshot displays the SQL Performance Analyzer Task Result: SYS.SPA\_JFV1 Create Baseline page:

**SQL Performance Analyzer Task Report: SYS.SPA\_JFV1** Cancel OK

**Create SQL Plan Baselines**

SQL Plan Baselines enable the optimizer to avoid performance regressions by requiring new plans to be at least as good as the better plans found in SQL trial 1.

**Regressed New Plan SQL Statements**

SQL ID	Net Impact on Workload (%)	Elapsed Time		Net Impact on SQL (%)	% of Workload	
		INITIAL_SQL_TRIAL	SECOND_SQL_TRIAL		INITIAL_SQL_TRIAL	SECOND_SQL_TRIAL
gkc6w4zhaysbk	-21.950	99.152	165.800	-67.220		
cqm5u0m6b68k5	-1.570	3.721	8.499	-128.430		
7866641pah2zg	-1.480	4.091	8.582	-109.750		

**Job Parameters**

Job Name:

Description:

**Schedule**

☒ Immediately  
☐ Later

Time Zone:

Date:

(example: Feb 17, 2010)

Time:    ☒ AM ☐ PM

2. Using Enterprise Manager, set the USE\_SQL\_PLAN\_BASELINES parameter.

Step	Page	Action	
a	Database home	Click the Server tab.	
b	Server	Click SQL Plan Control in the Query Optimizer section.	
c	SQL Plan Control	Click the SQL Plan Baseline tab.	
d	SQL Baselines subpage	If	Use SQL Plan Baselines is set to TRUE.
		Then	Go to step 3.
		Else	Click the FALSE link for Use SQL Plan Baselines.

## Practice 13-1: Seeding SQL Plan Baselines from SQL Performance Advisor (continued)

		Click the Database tab.
e	Initialization Parameters	Set optimizer_use_sql_plan_baselines to TRUE.
		Click OK.

- Use Enterprise Manager to create a new SQL Performance Analyzer session to look at the difference between the previous 11g trial and a new one that takes into account the previously created baselines.

**Note:** Screenshots of some of the steps described in the table are provided, as noted in the Page column, below the table. Refer to these screenshots as necessary.

Step	Page	Action		
a	Database home	Click the Software and Support tab.		
b	Software and Support	Click SQL Performance Analyzer.		
c	SQL Performance Analyzer	Click the SPA_JFV1 link.		
d	SQL Performance Analyzer Task: SYS.SPA_JFV1	Click Create SQL Trial.		
e	Create SQL Trial (See the example screenshot, which follows the table.)	Set values.	Field	Value
			SQLTrial Name	after_baselines
			Trial environment established check box	Select (on the lower right of the page)
			Per-SQL Time Limit	Unlimited
			Schedule	Immediately
		Click Submit.		
f	SQL Performance Analyzer Task: SYS.SPA_JFV1	Click Refresh until you see COMPLETED in the Status column for the AFTER_BASELINES trial.		
		Click Run SQL Trial Comparison.		

## Practice 13-1: Seeding SQL Plan Baselines from SQL Performance Advisor (continued)

g	Run SQL Trial Comparison  (See the example screenshot, which follows the table.)	Set values.	Field	Value
			Trial 1 Name	INITIAL_SQL_TRIAL
			Trial 2 Name	AFTER_BASELINES
			Comparison Metric	Elapsed Time
			Schedule	Immediately
	Click Submit.			
h	SQL Performance Analyzer Task: SYS.SPA_JFV1	You should see a Confirmation message, and a new line in the SQL Trial Comparisons section of the page.		
		Click the corresponding eyeglass icon in the Comparison Report column.		
i	SQL Performance Analyzer Task Result: SYS.SPA_JFV1	You should no longer see any significant regressing statements, and all previously regressing statements should now be executing faster.		

### e) Create SQL Trial.

#### Create SQL Trial

SQL Trials capture execution performance of the SQL Tuning Set under a given optimizer environment.

SQL Performance Analyzer Task **SYS.SPA\_JFV1**

SQL Tuning Set **SYS.STS\_JFV**

\* SQL Trial Name

SQL Trial Description

Creation Method

Per-SQL Time Limit

☒ **TIP** Time limit is on elapsed time of test execution of SQL

#### Schedule

Time Zone

☒ Immediately

☐ Later

Date   
(example: Feb 17, 2010)

Time  :  :  AM ☒ PM

#### Trial environment determines results

The SQL Tuning Set remains constant under the SQL Performance Analyzer Task and its SQL is executed in isolation to create each SQL Trial. Performance differences between trials are thus attributed to environmental differences between trials.

Environmental changes affecting SQL optimization and performance may need to be made manually prior to execution of the Trial. These could include changing initialization parameters, gathering or setting optimizer statistics and creating indexes.

The Creation Method determines how the SQL Trial is created and what contents are generated, as follows:

- Executing SQLs generates both plans and statistics by actually running the SQL statements.
- Generating plans invokes the optimizer to create execution plans only without running the SQL statements.
- Remote execution and plan generation are done over a public database link on the remote system.

**NOTE: Be sure trial environment has been established prior to submitting.**

☒ Trial environment established

Cancel

Submit

## Practice 13-1: Seeding SQL Plan Baselines from SQL Performance Advisor (continued)

g) Run SQL Trial Comparison.

### Run SQL Trial Comparison

Task Name

SYS.SPA\_JFV1

SQL Tuning Set

SYS.STS\_JFV

Trial 1 Name

INITIAL\_SQL\_TRIAL

Description

parameter optimizer\_features\_enable set to '11.2.0.1'

SQL Executed

Yes

Trial 2 Name

AFTER\_BASELINES

Description

SQL Executed

Yes

Comparison Metric

Elapsed Time

Compare trials to assess change impact

SQL Performance Analyzer trial comparison allows you to assess the impact on SQL Tuning Set performance of changes made between two trials.

It is important to know the difference between Trial 1 and Trial 2 execution environments in order to properly assign impacts to the changes between trials. Tracking environmental changes between trials is currently a user responsibility.

The selected comparison metric is used as the basis for comparison, and defaults to execute elapsed time when both trials contain test execution statistics. When execution statistics are not available, a less accurate comparison can be made using optimizer cost.

Schedule

Time Zone

Pacific/Pago\_Pago

☒ Immediately
 ☐ Later

Date

Feb 17, 2010

(example: Feb 17, 2010)

Time

6

58

05

☒ AM
 ☐ PM

4. Use Enterprise Manager to delete the created baselines.

Step	Page	Action
a	Database home	Click the Server tab.
b	Server	Click the SQL Plan Control link in the Query Optimizer section.
c	SQL Plan Control	Click the SQL Plan Baseline tab.
d	SQL Plan Baseline subpage	Select all baselines.
		Click Drop.
e	Confirmation	Click Yes.

5. In a terminal window, run the `./cleanup 13 1` script.

```
$ cd $HOME/workshops
$ ./cleanup 13 1
```

## Practice 13-2: SQL Plan Management (SPM)

In this practice, you see all phases of using SQL Plan Management.

1. Before you can start this practice, you must set up a new user. Execute the `spm_setup.sh` script in the `$HOME/labs/SPM` directory to set up the environment for this practice. This script creates the SPM user that you use throughout the practice.

```
$ cd /home/oracle/labs/SPM
$ cat ./spm_setup.sh

#!/bin/bash

sqlplus /nolog <<FIN!
connect / as sysdba

set echo on

drop user spm cascade;

create user spm identified by spm;

grant dba to spm;

alter system set SHARED_POOL_SIZE = 220M SCOPE = MEMORY;

alter system flush shared_pool;

FIN!

$ ./spm_setup.sh
```

### Automatic Plan Capture

2. The first component of SPM is Plan Capture. There are two main ways to capture plans: automatically (at run time) or bulk load. In this practice, you turn on automatic plan capture so that the SPM repository is automatically populated for any repeatable SQL statement. Turn on automatic plan capture by setting the `optimizer_capture_sql_plan_baselines` initialization parameter to `TRUE` in your `SQL*Plus` session, connected as the SPM user. After you have connected in your session, do not disconnect.

```
$ cd /home/oracle/labs/SPM

$ sqlplus spm/spm
```



**Practice 13-2: SQL Plan Management (SPM) (continued)**

```
SQL> show parameter baseline
```

NAME	TYPE	VALUE
optimizer_capture_sql_plan_baselines	boolean	FALSE
optimizer_use_sql_plan_baselines	boolean	FALSE

```
SQL> alter session set optimizer_capture_sql_plan_baselines = TRUE;
```

```
Session altered.
```

```
SQL> alter session set optimizer_use_sql_plan_baselines = TRUE;
```

```
Session altered.
```

```
SQL>
```

3. Execute the following query in your SQL\*Plus session (**Note:** There are no spaces in /\*LOAD...):

```
select /*LOAD_AUTO*/ * from sh.sales
where quantity_sold > 40 order by prod_id;
```

Use the query1.sql script to execute the query.

```
SQL> @query1.sql
```

```
SQL> select /*LOAD_AUTO*/ * from sh.sales
2> where quantity_sold > 40 order by prod_id;
```

```
no rows selected
```

```
SQL>
```

4. Because this is the first time that you have seen this SQL statement, it is not yet repeatable, so there is no plan baseline for it. To confirm this, check whether there are any plan baselines that exist for your statement. (Use the check\_baselines.sql script.)

```
-- Should not return any rows
```

```
SQL> @check_baselines.sql
```

```
SQL> set echo on
```

```
SQL>
```

```
SQL> select signature, sql_handle, sql_text, plan_name,
2         origin, enabled, accepted, fixed, autopurge
3 from dba_sql_plan_baselines
4 where sql_text like 'select /*LOAD_AUTO*/%';
```

```
no rows selected
```

## Practice 13-2: SQL Plan Management (SPM) (continued)

5. Execute the query that you ran in step 3.

```
SQL> @query1.sql
SQL> set echo on
SQL>
SQL> select /*LOAD_AUTO*/ * from sh.sales
      2  where quantity_sold > 40 order by prod_id;

no rows selected

SQL>
```

6. The SQL statement is now known to be repeatable and a plan baseline is automatically captured. Check whether the plan baseline was loaded for the previous statement. What do you observe?

You can see from the output that a baseline has been created and enabled for this SQL statement. You can also tell that this plan was captured automatically by checking the values of the `ORIGIN` column. (Use the `check_baselines.sql` script.)

```
-- You should see one accepted entry

SQL> @check_baselines.sql
SQL> set echo on
SQL>
SQL> select signature, sql_handle, sql_text, plan_name,
      2         origin, enabled, accepted, fixed, autopurge
      3  from dba_sql_plan_baselines
      4  where sql_text like 'select /*LOAD_AUTO*/%';

SIGNATURE SQL_HANDLE
-----
SQL_TEXT
-----
PLAN_NAME                                ORIGIN                                ENA ACC FIX AUT
-----
8.0622E+18 SYS_SQL_6fe28d438dfc352f
select /*LOAD_AUTO*/ * from sh.sales where quantity_sold > 40
order by prod_id
SYS_SQL_PLAN_8dfc352f54bc8843  AUTO-CAPTURE    YES YES NO  YES

SQL>
```

7. Change or alter the optimizer mode to use `FIRST_ROWS` optimization and execute your statement. Describe what happens.

## Practice 13-2: SQL Plan Management (SPM) (continued)

- a) This triggers the SQL statement to execute with a different plan.

```
SQL> alter session set optimizer_mode = first_rows;

Session altered.

SQL> @query1.sql
SQL> set echo on
SQL>
SQL> select /*LOAD_AUTO*/ * from sh.sales
      2  where quantity_sold > 40 order by prod_id;

no rows selected

SQL>
```

- b) Because the SQL statement will have a new plan, another plan baseline is automatically captured. You can confirm this by checking the plan baseline again.

```
SQL> @check_baselines.sql
SQL> set echo on
SQL>
SQL> select signature, sql_handle, sql_text, plan_name,
      2      origin, enabled, accepted, fixed, autopurge
      3  from dba_sql_plan_baselines
      4  where sql_text like 'select /*LOAD_AUTO*/%';

SIGNATURE SQL_HANDLE
-----
SQL_TEXT
-----
PLAN_NAME                                ORIGIN                                ENA ACC FIX AUT
-----
8.0622E+18 SYS_SQL_6fe28d438dfc352f
select /*LOAD_AUTO*/ * from sh.sales where quantity_sold > 40
order by prod_id
SYS_SQL_PLAN_8dfc352f11df68d0  AUTO-CAPTURE    YES NO  NO  YES

8.0622E+18 SYS_SQL_6fe28d438dfc352f
select /*LOAD_AUTO*/ * from sh.sales where quantity_sold > 40
order by prod_id
SYS_SQL_PLAN_8dfc352f54bc8843  AUTO-CAPTURE    YES YES NO  YES
```

- c) Now you see two plan baselines for your query, but notice that the new plan has not been accepted. This new plan will have to be validated before it is acceptable as a good plan to use.
8. Reset the optimizer mode to the default values and disable autocapture of plan baselines.

**Practice 13-2: SQL Plan Management (SPM) (continued)**

```
SQL> alter session set optimizer_mode = all_rows;

Session altered.

SQL> alter session set optimizer_capture_sql_plan_baselines =
FALSE;

Session altered.
```

9. Purge the plan baselines and confirm that the SQL plan baseline is empty. Use `purge_auto_baseline.sql`.

```
SQL> @purge_auto_baseline.sql
SQL> set echo on
SQL>
SQL> variable cnt number;
SQL>
SQL> exec :cnt :=
dbms_spm.drop_sql_plan_baseline('SYS_SQL_6fe28d438dfc352f');

PL/SQL procedure successfully completed.

SQL>
SQL> select signature, sql_handle, sql_text, plan_name,
2         origin, enabled, accepted, fixed, autopurge
3   from dba_sql_plan_baselines
4  where sql_text like 'select /*LOAD_AUTO*/%';

no rows selected

SQL>
```

**Loading Plans from SQL Tuning Sets**

10. Still connected to your SQL\*Plus session, check the execution plan for the following SQL statement, and then execute it (use `explain_query2.sql` and `query2.sql`).

```
select /*LOAD_STS*/ * from sh.sales
where quantity_sold > 40 order by prod_id;
```

```
SQL> @explain_query2.sql

-- You should see a Full Table Scan

SQL> set echo on
SQL>
SQL> explain plan for
2   select /*LOAD_STS*/ * from sh.sales
3   where quantity_sold > 40 order by prod_id;

Explained.
```

**Practice 13-2: SQL Plan Management (SPM) (continued)**

```

SQL>
SQL> select * from
table(dbms_xplan.display(null,null,'basic'));

PLAN_TABLE_OUTPUT
-----
Plan hash value: 3803407550

-----
| Id | Operation                                | Name |
-----
|  0 | SELECT STATEMENT                        |      |
|  1 |   SORT ORDER BY                        |      |
|  2 |    PARTITION RANGE ALL                 |      |
|  3 |     TABLE ACCESS FULL                 | SALES |
-----

10 rows selected.

SQL> @query2.sql
SQL> set echo on
SQL>
SQL> select /*LOAD_STS*/ * from sh.sales
      2  where quantity_sold > 40 order by prod_id;

no rows selected

```

11. Alter the optimizer mode to use FIRST\_ROWS optimization, check the execution plan, and execute the query.

```

SQL> alter session set optimizer_mode = first_rows;

Session altered.

-- You should see a different plan
-- (Table Access By Local Index)

SQL> @explain_query2.sql
SQL> set echo on
SQL>
SQL> explain plan for
      2  select /*LOAD_STS*/ * from sh.sales
      3  where quantity_sold > 40 order by prod_id;

Explained.

SQL>
SQL> select * from
table(dbms_xplan.display(null,null,'basic'));

PLAN_TABLE_OUTPUT

```

**Practice 13-2: SQL Plan Management (SPM) (continued)**

```
-----
Plan hash value: 899219946
-----
```

Id	Operation	Name
0	SELECT STATEMENT	
1	SORT ORDER BY	
2	PARTITION RANGE ALL	
3	TABLE ACCESS BY LOCAL INDEX ROWID	SALES
4	BITMAP CONVERSION TO ROWIDS	
5	BITMAP INDEX FULL SCAN	SALES_PROMO_BIX

```
-----
PLAN_TABLE_OUTPUT
-----
```

```
12 rows selected.
```

```
SQL>
```

```
SQL> @query2.sql
```

```
SQL> set echo on
```

```
SQL>
```

```
SQL> select /*LOAD_STS*/ * from sh.sales
       2  where quantity_sold > 40 order by prod_id;
```

```
no rows selected
```

```
SQL>
```

12. Reset the optimizer mode to ALL\_ROWS optimization.

```
SQL> alter session set optimizer_mode = all_rows;
```

```
Session altered.
```

13. Verify that there are no baseline plans for your statement (use the check\_baselines2.sql script).

```
SQL> @check_baselines2.sql
```

```
SQL> set echo on
```

```
SQL>
```

```
SQL> select signature, sql_handle, sql_text, plan_name,
       2      origin, enabled, accepted, fixed, autopurge
       3  from dba_sql_plan_baselines
       4  where sql_text like 'select /*LOAD_STS*/%';
```

```
no rows selected
```

```
SQL>
```

14. Create a SQL tuning set that captures the SELECT statements that contain the LOAD\_STS hint. These statements are in the cursor cache. The STS should be called SPM\_STS and owned by the SPM user. Use the catchup\_sts.sql script to capture these statements.

**Practice 13-2: SQL Plan Management (SPM) (continued)**

```

SQL> @catchup_sts.sql
SQL> set echo on
SQL>
SQL> exec sys.dbms_sqltune.create_sqlset(-
> sqlset_name => 'SPM_STS', sqlset_owner => 'SPM');

PL/SQL procedure successfully completed.

SQL>
SQL> DECLARE
  2     stscur    dbms_sqltune.sqlset_cursor;
  3 BEGIN
  4     OPEN stscur FOR
  5         SELECT VALUE(P)
  6         FROM   TABLE(dbms_sqltune.select_cursor_cache(
  7             'sql_text like ''select /*LOAD_STS*/%'',
  8             null, null, null, null, null, null, 'ALL')) P;
  9
 10     -- populate the sqlset
 11     dbms_sqltune.load_sqlset(sqlset_name      => 'SPM_STS',
 12                             populate_cursor => stscur,
 13                             sqlset_owner    => 'SPM');
 14 END;
 15 /

PL/SQL procedure successfully completed.

SQL>

```

15. Verify which SQL statements are in SPM\_STS. (Use the check\_sts.sql script.)

SPM\_STS has your SQL statement with two different plans.

```

SQL> @check_sts.sql
SQL> set echo on
SQL>
SQL> select sql_text from dba_sqlset_statements
  2  where sqlset_name='SPM_STS';

SQL_TEXT
-----
select /*LOAD_STS*/ * from sh.sales
where quantity_sold > 40 order by prod_id

select /*LOAD_STS*/ * from sh.sales
where quantity_sold > 40 order by prod_id

SQL>

```

**Practice 13-2: SQL Plan Management (SPM) (continued)**

16. Populate the plan baseline repository with the plans found in SPM\_STS. Use the `populate_baseline.sql` script for that:

```
SQL> @populate_baseline.sql
SQL> set echo on
SQL>
SQL> variable cnt number;
SQL>
SQL> exec :cnt := dbms_spm.load_plans_from_sqlset( -
>         sqlset_name => 'SPM_STS', -
>         basic_filter => 'sql_text like 'select
/*LOAD_STS*/%'');

PL/SQL procedure successfully completed.
SQL>
```

17. Confirm that the plan baselines are loaded and note the value in the origin column. What do you observe? (Use `check_baselines2.sql`.)

You should see `MANUAL-LOAD` because you manually loaded these plans. Also note that this time, both plans are accepted.

```
SQL> @check_baselines2.sql
SQL> set echo on
SQL>
SQL> select signature, sql_handle, sql_text, plan_name,
2         origin, enabled, accepted, fixed, autopurge
3        from dba_sql_plan_baselines
4        where sql_text like 'select /*LOAD_STS*/%';

SIGNATURE SQL_HANDLE
-----
SQL_TEXT
-----
PLAN_NAME                                ORIGIN                                ENA ACC FIX AUT
-----
1.2134E+19 SYS_SQL_a8632bd857a4a25e
select /*LOAD_STS*/ * from sh.sales
where quantity_sold > 40 order by prod_id
SYS_SQL_PLAN_57a4a25e11df68d0  MANUAL-LOAD  YES YES NO  YES
1.2134E+19 SYS_SQL_a8632bd857a4a25e
select /*LOAD_STS*/ * from sh.sales
where quantity_sold > 40 order by prod_id
SYS_SQL_PLAN_57a4a25e54bc8843  MANUAL-LOAD  YES YES NO  YES

SQL>
```



**Practice 13-2: SQL Plan Management (SPM) (continued)**

18. Purge the plan baselines and drop SPM\_STS. Use the `purge_sts_baseline.sql` script:

```
SQL> @purge_sts_baseline.sql
SQL> set echo on
SQL>
SQL> variable cnt number;
SQL>
SQL> exec :cnt := dbms_spm.drop_sql_plan_baseline(-
>             'SYS_SQL_a8632bd857a4a25e');

PL/SQL procedure successfully completed.

SQL>
SQL> print cnt;

          CNT
-----
          2

SQL>
SQL> select signature, sql_handle, sql_text, plan_name,
2         origin, enabled, accepted, fixed, autopurge
3   from dba_sql_plan_baselines
4  where sql_text like 'select /*LOAD_STS*/%';

no rows selected

SQL>
SQL> exec sys.dbms_sqltune.drop_sqlset(-
>             sqlset_name  => 'SPM_STS', -
>             sqlset_owner => 'SPM');

PL/SQL procedure successfully completed.

SQL>
```

**Loading Plans from the Cursor Cache**

19. Now, you see how to directly load plan baselines from the cursor cache. Before you begin, you need some SQL statements. Still connected to your SQL\*Plus session, check the execution plan for the following SQL statement, and then execute it. (Use the `explain_query3.sql` and `query3.sql` scripts.)

```
select /*LOAD_CC*/ * from sh.sales
where quantity_sold > 40 order by prod_id;
```

```
SQL> @explain_query3.sql
SQL> set echo on
SQL>
```

**Practice 13-2: SQL Plan Management (SPM) (continued)**

```
SQL> explain plan for
  2 select /*LOAD_CC*/ * from sh.sales
  3 where quantity_sold > 40 order by prod_id;
```

Explained.

```
SQL>
SQL> select * from
table(dbms_xplan.display(null,null,'basic'));
```

PLAN\_TABLE\_OUTPUT

-----

Plan hash value: 3803407550

Id	Operation	Name
0	SELECT STATEMENT	
1	SORT ORDER BY	
2	PARTITION RANGE ALL	
3	TABLE ACCESS FULL	SALES

-----

10 rows selected.

```
SQL>
SQL> @query3.sql
SQL> set echo on
SQL>
SQL> select /*LOAD_CC*/ * from sh.sales
  2 where quantity_sold > 40 order by prod_id;
```

no rows selected

SQL>

20. Now change the optimizer mode to use FIRST\_ROWS optimization and execute the same script as in the previous step. What do you observe?

a) You should see a different execution plan.

```
SQL> alter session set optimizer_mode = first_rows;
```

Session altered.

```
SQL> @explain_query3.sql
SQL> set echo on
SQL>
SQL> explain plan for
  2 select /*LOAD_CC*/ * from sh.sales
  3 where quantity_sold > 40 order by prod_id;
```

**Practice 13-2: SQL Plan Management (SPM) (continued)**

Explained.

SQL>

```
SQL> select * from
table(dbms_xplan.display(null,null,'basic'));
```

PLAN\_TABLE\_OUTPUT

-----  
Plan hash value: 899219946  
-----

Id	Operation	Name
0	SELECT STATEMENT	
1	SORT ORDER BY	
2	PARTITION RANGE ALL	
3	TABLE ACCESS BY LOCAL INDEX ROWID	SALES
4	BITMAP CONVERSION TO ROWIDS	
5	BITMAP INDEX FULL SCAN	SALES_PROMO_BIX

PLAN\_TABLE\_OUTPUT

-----  
12 rows selected.

SQL>

```
SQL> @query3.sql
```

```
SQL> set echo on
```

SQL>

```
SQL> select /*LOAD_CC*/ * from sh.sales
       2  where quantity_sold > 40 order by prod_id;
```

no rows selected

SQL>

21. Reset the optimizer mode to ALL\_ROWS.

```
SQL> alter session set optimizer_mode = all_rows;
```

Session altered.

22. Now that the cursor cache is populated, you must get the SQL ID for your SQL statement. Use the SQL ID to filter the content of the cursor cache and load the baselines with these two plans. Use the load\_cc\_baseline.sql script.

```
SQL> @load_cc_baseline.sql
```

```
SQL> set echo on
```

SQL>

```
SQL> variable cnt number;
```

SQL>

```
SQL> variable sqlid varchar2(20);
```

SQL>

**Practice 13-2: SQL Plan Management (SPM) (continued)**

```

SQL> begin
  2   select distinct sql_id into :sqlid from v$sql
  3   where sql_text like 'select /*LOAD_CC*/%';
  4 end;
  5 /

PL/SQL procedure successfully completed.

SQL>
SQL> print sqlid;

SQLID
-----
6qc9wxhgzzz8g

SQL>
SQL> exec :cnt := dbms_spm.load_plans_from_cursor_cache(-
>               sql_id => :sqlid);

PL/SQL procedure successfully completed.

SQL>

```

23. Confirm that the baselines were loaded. (Use check\_baselines3.sql.)

```

SQL> @check_baselines3.sql
SQL> set echo on
SQL>
SQL> select signature, sql_handle, sql_text, plan_name,
  2         origin, enabled, accepted, fixed, autopurge
  3   from dba_sql_plan_baselines
  4   where sql_text like 'select /*LOAD_CC*/%';

SIGNATURE SQL_HANDLE
-----
SQL_TEXT
-----
PLAN_NAME                                ORIGIN                                ENA ACC FIX AUT
-----
1.7783E+19 SYS_SQL_f6cb7f742ef93547
select /*LOAD_CC*/ * from sh.sales
where quantity_sold > 40 order by prod_id
SYS_SQL_PLAN_2ef9354711df68d0  MANUAL-LOAD  YES YES NO  YES

1.7783E+19 SYS_SQL_f6cb7f742ef93547
select /*LOAD_CC*/ * from sh.sales
where quantity_sold > 40 order by prod_id
SYS_SQL_PLAN_2ef9354754bc8843  MANUAL-LOAD  YES YES NO  YES

-- You should see two accepted baselines

```

**Practice 13-2: SQL Plan Management (SPM) (continued)**

24. Purge the plan baselines.

```
SQL> @purge_cc_baseline.sql
SQL> set echo on
SQL>
SQL> variable cnt number;
SQL>
SQL> exec :cnt := dbms_spm.drop_sql_plan_baseline(-
>                'SYS_SQL_f6cb7f742ef93547');

PL/SQL procedure successfully completed.

SQL>
SQL> print cnt;

          CNT
-----
          2

SQL>
SQL> REM Check that plan baselines were purged:
SQL>
SQL> select signature, sql_handle, sql_text, plan_name,
2          origin, enabled, accepted, fixed, autopurge
3   from dba_sql_plan_baselines
4   where sql_text like 'select /*LOAD_CC*/%';

no rows selected

SQL>
```

**Optimizer Plan Selection**

25. Now that you know how to capture plans, look at how the optimizer selects which plan baselines to use. First, you create two baselines for a statement and show them being used. Then you disable one of the baselines and see the second baseline being used. Finally, you disable both baselines and show that now the optimizer falls back to the default behavior of a cost-based plan. Start by executing the same query twice, with different plans. Determine the execution of the following statement, and then execute it. (Use the `explain_query4.sql` and `query4.sql` scripts.)

```
select /*SPM_USE*/ * from sh.sales
where quantity_sold > 40 order by prod_id
```

```
SQL> @explain_query4.sql
SQL> set echo on
SQL>
SQL> explain plan for
2   select /*SPM_USE*/ * from sh.sales
```

**Practice 13-2: SQL Plan Management (SPM) (continued)**

```
3 where quantity_sold > 40 order by prod_id;
```

Explained.

```
SQL>
```

```
SQL> select * from
table(dbms_xplan.display(null,null,'basic'));
```

```
PLAN_TABLE_OUTPUT
```

```
-----
Plan hash value: 3803407550
```

```
-----
| Id | Operation | Name |
-----
| 0 | SELECT STATEMENT |      |
| 1 |   SORT ORDER BY |      |
| 2 |     PARTITION RANGE ALL |      |
| 3 |       TABLE ACCESS FULL | SALES |
-----
```

```
10 rows selected.
```

```
SQL>
```

```
SQL> @query4.sql
```

```
SQL> set echo on
```

```
SQL>
```

```
SQL> select /*SPM_USE*/ * from sh.sales
2 where quantity_sold > 40 order by prod_id;
```

```
no rows selected
```

```
SQL>
```

26. Change the optimizer mode to use FIRST\_ROWS optimization and execute the same script as in the previous step. What do you observe?

You should see a different execution plan.

```
SQL> alter session set optimizer_mode = first_rows;
```

```
Session altered.
```

```
SQL> @explain_query4.sql
```

```
SQL> set echo on
```

```
SQL>
```

```
SQL> explain plan for
2 select /*SPM_USE*/ * from sh.sales
3 where quantity_sold > 40 order by prod_id;
```

Explained.

```
SQL>
```

**Practice 13-2: SQL Plan Management (SPM) (continued)**

```
SQL> select * from
table(dbms_xplan.display(null,null,'basic'));
```

```
PLAN_TABLE_OUTPUT
```

```
-----
Plan hash value: 899219946
-----
```

Id	Operation	Name
0	SELECT STATEMENT	
1	SORT ORDER BY	
2	PARTITION RANGE ALL	
3	TABLE ACCESS BY LOCAL INDEX ROWID	SALES
4	BITMAP CONVERSION TO ROWIDS	
5	BITMAP INDEX FULL SCAN	SALES_PROMO_BIX

```
PLAN_TABLE_OUTPUT
```

```
-----
12 rows selected.
```

```
SQL>
```

```
SQL> @query4.sql
```

```
SQL> set echo on
```

```
SQL>
```

```
SQL> select /*SPM_USE*/ * from sh.sales
       2  where quantity_sold > 40 order by prod_id;
```

```
no rows selected
```

```
SQL>
```

27. Reset the optimizer mode to ALL\_ROWS.

```
SQL> alter session set optimizer_mode = all_rows;
```

```
Session altered.
```

28. Populate the baseline with the two plans for your statement directly from the cursor cache. Use the load\_use\_baseline.sql script for that. Then verify that baselines were loaded. What do you observe?

You should see both plan baselines loaded. Note that both plans have been marked acceptable. This is because both plans were present in the cursor cache at the time of the load, and because these plans have been manually loaded, it is assumed that both plans have acceptable performance.

```
SQL> @load_use_baseline.sql
```

```
SQL> set echo on
```

```
SQL>
```

```
SQL> variable cnt number;
```

**Practice 13-2: SQL Plan Management (SPM) (continued)**

```

SQL>
SQL> variable sqlid varchar2(20);
SQL>
SQL> begin
  2   select distinct sql_id into :sqlid from v$sql
  3     where sql_text like 'select /*SPM_USE*/%';
  4 end;
  5 /

PL/SQL procedure successfully completed.

SQL>
SQL> print sqlid;

SQLID
-----
2pma6tcaczdc8

SQL>
SQL> exec :cnt := dbms_spm.load_plans_from_cursor_cache(-
>               sql_id => :sqlid);

PL/SQL procedure successfully completed.

SQL>
SQL> print cnt;

          CNT
-----
          2

SQL>
SQL> select signature, sql_handle, sql_text, plan_name,
  2         origin, enabled, accepted, fixed, autopurge
  3   from dba_sql_plan_baselines
  4  where sql_text like 'select /*SPM_USE*/%';

SIGNATURE SQL_HANDLE
-----
SQL_TEXT
-----
PLAN_NAME                                ORIGIN                                ENA ACC FIX AUT
-----
7.6492E+17 SYS_SQL_0a9d872600ece455
select /*SPM_USE*/ * from sh.sales
where quantity_sold > 40 order by prod_id
SYS_SQL_PLAN_00ece45511df68d0  MANUAL-LOAD  YES YES NO  YES

7.6492E+17 SYS_SQL_0a9d872600ece455
select /*SPM_USE*/ * from sh.sales
where quantity_sold > 40 order by prod_id
SYS_SQL_PLAN_00ece45554bc8843  MANUAL-LOAD  YES YES NO  YES

SQL>

```



**Practice 13-2: SQL Plan Management (SPM) (continued)**

29. Determine the baseline and the execution plan used to execute the following query:

```
select /*SPM_USE*/ * from sh.sales
where quantity_sold > 40 order by prod_id.
```

What do you observe?

The note at the end of the explain output tells you that the system is using a baseline. From the execution plan, you can see that you are using the first baseline, a full-table scan. Use the `explain_query4_note.sql` script.

```
SQL> @explain_query4_note.sql
SQL> set echo on
SQL>
SQL> explain plan for
  2  select /*SPM_USE*/ * from sh.sales
  3  where quantity_sold > 40 order by prod_id;

Explained.

SQL>
SQL> select * from table(dbms_xplan.display(null,null,'basic
+note'));
```

PLAN\_TABLE\_OUTPUT

---

Plan hash value: 3803407550

Id	Operation	Name
0	SELECT STATEMENT	
1	SORT ORDER BY	
2	PARTITION RANGE ALL	
3	TABLE ACCESS FULL	SALES

---

PLAN\_TABLE\_OUTPUT

---

Note

---

- SQL plan baseline "SYS\_SQL\_PLAN\_00ece45554bc8843" used for this statement

14 rows selected.

SQL>

**Practice 13-2: SQL Plan Management (SPM) (continued)**

30. Disable that plan baseline, and check whether the system uses the other plan baseline when executing the statement again. Use the `check_baseline_used.sql` script for that. What do you observe?
- a) Now from the execution plan, you see that you are using an index scan instead of a full-table scan. So this is the second baseline.

```
SQL> @check_baseline_used.sql
SQL> set echo on
SQL>
SQL> variable cnt number;
SQL>
SQL> select sql_handle,plan_name
       2  from dba_sql_plan_baselines
       3  where sql_text like 'select /*SPM_USE*/%';

SQL_HANDLE                                PLAN_NAME
-----
SYS_SQL_0a9d872600ece455                 SYS_SQL_PLAN_00ece45511df68d0
SYS_SQL_0a9d872600ece455                 SYS_SQL_PLAN_00ece45554bc8843

SQL>
SQL>
SQL> exec :cnt := dbms_spm.alter_sql_plan_baseline( -
>         sql_handle      => 'SYS_SQL_0a9d872600ece455', -
>         plan_name       => 'SYS_SQL_PLAN_00ece45554bc8843', -
>         attribute_name  => 'ENABLED', -
>         attribute_value => 'NO');

PL/SQL procedure successfully completed.

SQL>
SQL> select signature, sql_handle, sql_text, plan_name,
       2         origin, enabled, accepted, fixed, autopurge
       3  from dba_sql_plan_baselines
       4  where sql_text like 'select /*SPM_USE*/%';

SIGNATURE SQL_HANDLE
-----
SQL_TEXT
-----
PLAN_NAME                                ORIGIN      ENA ACC FIX AUT
-----
7.6492E+17 SYS_SQL_0a9d872600ece455
select /*SPM_USE*/ * from sh.sales
where quantity_sold > 40 order by prod_id
SYS_SQL_PLAN_00ece45511df68d0  MANUAL-LOAD  YES YES NO  YES

7.6492E+17 SYS_SQL_0a9d872600ece455
select /*SPM_USE*/ * from sh.sales
```

**Practice 13-2: SQL Plan Management (SPM) (continued)**

```

where quantity_sold > 40 order by prod_id
SYS_SQL_PLAN_00ece45554bc8843  MANUAL-LOAD      NO  YES NO  YES

SQL>
SQL>
SQL> explain plan for select /*SPM_USE*/ * from sh.sales
      2          where quantity_sold > 40 order by prod_id;

Explained.

SQL>
SQL> select * from table(dbms_xplan.display(null, null, 'basic
+note'));

PLAN_TABLE_OUTPUT
-----
Plan hash value: 899219946

-----
| Id | Operation                                | Name |
-----|-----|-----|
|  0 | SELECT STATEMENT                        |      |
|  1 |   SORT ORDER BY                        |      |
|  2 |     PARTITION RANGE ALL                 |      |
|  3 |       TABLE ACCESS BY LOCAL INDEX ROWID | SALES |
|  4 |         BITMAP CONVERSION TO ROWIDS     |      |
|  5 |           BITMAP INDEX FULL SCAN        | SALES_PROMO_BIX |

PLAN_TABLE_OUTPUT
-----

Note
-----
- SQL plan baseline "SYS_SQL_PLAN_00ece45511df68d0" used
for this statement

16 rows selected.

SQL>

```

31. Disable this other plan baseline and check whether the system falls back to the cost-based approach when executing the explain plan for the statement. Use the `check_baseline_used2.sql` script.

You know that the optimizer has gone back to the default cost-based approach because there is no note at the end of the plan stating that a baseline was used.

```

SQL> @check_baseline_used2.sql
SQL> set echo on
SQL>

```

**Practice 13-2: SQL Plan Management (SPM) (continued)**

```
SQL> variable cnt number;
SQL>
SQL> exec :cnt := dbms_spm.alter_sql_plan_baseline( -
>         sql_handle          => 'SYS_SQL_0a9d872600ece455', -
>         plan_name           =>
'SYS_SQL_PLAN_00ece45511df68d0', -
>         attribute_name      => 'ENABLED', -
>         attribute_value     => 'NO');
```

PL/SQL procedure successfully completed.

```
SQL>
SQL> select signature, sql_handle, sql_text, plan_name,
2         origin, enabled, accepted, fixed, autopurge
3   from dba_sql_plan_baselines
4  where sql_text like 'select /*SPM_USE*/%';
```

SIGNATURE	SQL_HANDLE	SQL_TEXT	PLAN_NAME	ORIGIN	ENA	ACC	FIX	AUT
7.6492E+17	SYS_SQL_0a9d872600ece455	select /*SPM_USE*/ * from sh.sales where quantity_sold > 40 order by prod_id	SYS_SQL_PLAN_00ece45511df68d0	MANUAL-LOAD	NO	YES	NO	YES
7.6492E+17	SYS_SQL_0a9d872600ece455	select /*SPM_USE*/ * from sh.sales where quantity_sold > 40 order by prod_id	SYS_SQL_PLAN_00ece45554bc8843	MANUAL-LOAD	NO	YES	NO	YES

```
SQL>
SQL>
SQL> explain plan for select /*SPM_USE*/ * from sh.sales
2   where quantity_sold > 40 order by prod_id;
```

Explained.

```
SQL>
SQL> select * from table(dbms_xplan.display(null, null, 'basic
+note'));
```

PLAN\_TABLE\_OUTPUT

Plan hash value: 3803407550

Id	Operation	Name
----	-----------	------

**Practice 13-2: SQL Plan Management (SPM) (continued)**

0	SELECT STATEMENT		
1	SORT ORDER BY		
2	PARTITION RANGE ALL		
3	TABLE ACCESS FULL	SALES	

---

10 rows selected.

SQL>

32. Drop the plan baselines and check whether they are purged. Use the `purge_use_baseline.sql` script.

```
SQL> @purge_use_baseline
SQL> set echo on
SQL>
SQL> variable cnt number;
SQL>
SQL> exec :cnt := dbms_spm.drop_sql_plan_baseline(-
>              'SYS_SQL_0a9d872600ece455');

PL/SQL procedure successfully completed.

SQL>
SQL> print cnt;

          CNT
-----
          2

SQL>
SQL> select signature, sql_handle, sql_text, plan_name,
2         origin, enabled, accepted, fixed, autopurge
3   from dba_sql_plan_baselines
4  where sql_text like 'select /*SPM_USE*/%';

no rows selected

SQL>
```

33. One of the methods used to enable plan evolution (or plan verification) is Automatic SQL Tuning that is run as an automated task in a maintenance window. Automatic SQL Tuning targets only the high-load SQL statements; for them, it automatically implements actions such as making a successfully verified plan an accepted plan. Here, you manually trigger SQL tuning to find a better plan for a given SQL statement. First, determine the execution plan of the following statement:

```
select /*+ USE_NL(s c) FULL(s) FULL(c) */
       c.cust_id, sum(s.quantity_sold)
from sh.sales s, sh.customers c
```

## Practice 13-2: SQL Plan Management (SPM) (continued)

```
where s.cust_id = c.cust_id and c.cust_id < 2
group by c.cust_id
```

Some optimizer hints to the statements have been added to ensure that you get a less than optimal plan at first. Use the `check_evolve_plan.sql` script for that. What do you observe?

As you can see, the execution plan is being forced by the hints to perform two full-table scans, followed by a nest loop join.

```
SQL> @check_evolve_plan.sql
SQL> set echo on
SQL>
SQL> explain plan for
  2  select /*+ USE_NL(s c) FULL(s) FULL(c) */
  3         c.cust_id, sum(s.quantity_sold)
  4  from sh.sales s, sh.customers c
  5  where s.cust_id = c.cust_id and c.cust_id < 2
  6  group by c.cust_id;

Explained.

SQL>
SQL> select * from table(dbms_xplan.display(null, null));
```

PLAN\_TABLE\_OUTPUT

Plan hash value: 4005616876

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Pstart	Pstop
----	-----------	------	------	-------	-------------	------	--------	-------

PLAN\_TABLE\_OUTPUT

0	SELECT STATEMENT		1	39	893 (2)	00:00:11		
1	HASH GROUP BY		1	39	893 (2)	00:00:11		
2	NESTED LOOPS		1	31	892 (2)	00:00:11		
* 3	TABLE ACCESS FULL	CUSTOMERS	1	5	405 (1)	00:00:05		
4	PARTITION RANGE ALL		1	8	488 (2)	00:00:06	1	28
* 5	TABLE ACCESS FULL	SALES	1	8	488 (2)	00:00:06	1	28

PLAN\_TABLE\_OUTPUT

Predicate Information (identified by operation id):

```
3 - filter("C"."CUST_ID"<2)
5 - filter("S"."CUST_ID"<2 AND "S"."CUST_ID"="C"."CUST_ID")
```

18 rows selected.

SQL>

**Practice 13-2: SQL Plan Management (SPM) (continued)**

34. Now execute the statement so that you can get the plan in the cursor cache and load the corresponding plan baseline. Use the `load_evolve_baseline.sql` script for that. What do you observe?

You see that the current plan is both enabled and accepted, but not fixed.

```
SQL> @load_evolve_baseline
SQL> set echo on
SQL>
SQL> variable cnt number;
SQL>
SQL> variable sqlid varchar2(20);
SQL>
SQL> select /*+ USE_NL(s c) FULL(s) FULL(c) */
2         c.cust_id, sum(s.quantity_sold)
3   from sh.sales s, sh.customers c
4  where s.cust_id = c.cust_id and c.cust_id < 2
5  group by c.cust_id;

no rows selected

SQL>
SQL> begin
2   select sql_id into :sqlid from v$sql
3   where sql_text like 'select /*+ USE_NL(s c) FULL(s)
FULL(c) */%';
4 end;
5 /

PL/SQL procedure successfully completed.

SQL>
SQL> exec :cnt := dbms_spm.load_plans_from_cursor_cache(-
>               sql_id => :sqlid);

PL/SQL procedure successfully completed.

SQL>
SQL> select signature, sql_handle, sql_text, plan_name,
2         origin, enabled, accepted, fixed, autopurge
3   from dba_sql_plan_baselines
4  where sql_text like 'select /*+ USE_NL(s c) FULL(s)
FULL(c) */%';

SIGNATURE SQL_HANDLE
-----
SQL_TEXT
-----
PLAN_NAME                                ORIGIN                                ENA ACC FIX AUT
-----
1.7750E+18 SYS_SQL_18a1ef14c17f5b75
```

### ***Practice 13-2: SQL Plan Management (SPM) (continued)***

```
select /*+ USE_NL(s c) FULL(s) FULL(c) */ c.cust_id,  
sum(s.quantity_sold)  
from s  
SYS_SQL_PLAN_c17f5b75dc5f94e5    MANUAL-LOAD    YES YES NO    YES  
  
SQL>
```



**Practice 13-2: SQL Plan Management (SPM) (continued)**

35. Manually create and execute a SQL tuning task to tune your statement. Use the `tune_evolve_sql.sql` script.

```
SQL> @tune_evolve_sql.sql
SQL> set echo on
SQL>
SQL> variable sqltext varchar2(4000);
SQL>
SQL> BEGIN
  2   :sqltext := q'# select /*+ USE_NL(s c) FULL(s) FULL(c)*/
  3               c.cust_id, sum(s.quantity_sold)
  4               from sh.sales s, sh.customers c
  5               where s.cust_id = c.cust_id
  6                   and c.cust_id < 2
  7               group by c.cust_id
  8               #';
  9   END;
10   /

PL/SQL procedure successfully completed.

SQL>
SQL> variable spmtune   varchar2(30);
SQL>
SQL> exec :spmtune := dbms_sqltune.create_tuning_task(-
>               sql_text => :sqltext);

PL/SQL procedure successfully completed.

SQL>
SQL> exec dbms_sqltune.execute_tuning_task(:spmtune);

PL/SQL procedure successfully completed.

SQL>
```

36. Now that the tuning task has been completed, run the report and see what recommendations have been made for your statement. What do you observe?

There are two recommendations: a SQL profile or a new index creation. Use the `report_evolve_tuning.sql` script.

```
SQL> @report_evolve_tuning.sql
SQL> set echo on
SQL>
SQL> set long 10000
SQL>
SQL> select dbms_sqltune.report_tuning_task(:spmtune, 'TEXT')
  2   from dual;
-- Report Follows --
```

## Practice 13-2: SQL Plan Management (SPM) (continued)

37. Accept the SQL profile proposed by the SQL Tuning Advisor. Use the `accept_evolve_baseline.sql` script to accept the recommended SQL profile. What happens?

Accepting the profile causes a new SQL profile and plan baseline to be created for your statement. Now you see two baselines for your statement. Both of them are enabled and accepted.

**Note:** One is MANUAL-LOAD and the other is MANUAL-SQLTUNE.

```
SQL> @accept_evolve_baseline.sql
SQL> set echo on
SQL>
SQL> select signature, sql_handle, sql_text, plan_name,
2      origin, enabled, accepted, fixed, autopurge
3      from dba_sql_plan_baselines
4      where sql_text like
5          'select /*+ USE_NL(s c) FULL(s) FULL(c) */%';

SIGNATURE SQL_HANDLE
-----
SQL_TEXT
-----
PLAN_NAME                                ORIGIN          ENA ACC FIX AUT
-----
1.7750E+18 SYS_SQL_18a1ef14c17f5b75
select /*+ USE_NL(s c) FULL(s) FULL(c) */ c.cust_id,
sum(s.quantity_sold)
from sh.sales s, sh.customers c
where s.cust_id = c.cust_id and c.cust_id < 2
group by c.cust_id
SYS_SQL_PLAN_c17f5b75dc5f94e5  MANUAL-LOAD      YES YES NO  YES

SQL>
SQL> exec dbms_sqltune.accept_sql_profile(-
>      task_name => :spmtune,-
>      name => 'SPM_SQL_PROF');

PL/SQL procedure successfully completed.

SQL>
SQL> select signature, category, name, sql_text
2      from dba_sql_profiles where name like 'SPM%';

SIGNATURE CATEGORY                                NAME
-----
SQL_TEXT
-----
1.7750E+18 DEFAULT                                SPM_SQL_PROF
select /*+ USE_NL(s c) FULL(s) FULL(c) */ c.cust_id,
sum(s.quantity_sold)
```

**Practice 13-2: SQL Plan Management (SPM) (continued)**

```

from sh.sales s, sh.customers c
where s.cust_id = c.cust_id and c.cust_id < 2
group by c.cust_id

SQL> select signature, sql_handle, sql_text, plan_name,
       2  origin, enabled, accepted, fixed, autopurge
       3  from dba_sql_plan_baselines
       4  where sql_text like
       5         'select /*+ USE_NL(s c) FULL(s) FULL(c) */%';

SIGNATURE SQL_HANDLE
-----
SQL_TEXT
-----
PLAN_NAME                                ORIGIN                                ENA ACC FIX AUT
-----
1.7750E+18 SYS_SQL_18a1ef14c17f5b75
select /*+ USE_NL(s c) FULL(s) FULL(c) */ c.cust_id,
sum(s.quantity_sold)
from sh.sales s, sh.customers c
where s.cust_id = c.cust_id and c.cust_id < 2
group by c.cust_id
SYS_SQL_PLAN_c17f5b75a10c1dcf  MANUAL-SQLTUNE YES YES NO  YES

1.7750E+18 SYS_SQL_18a1ef14c17f5b75
select /*+ USE_NL(s c) FULL(s) FULL(c) */ c.cust_id,
sum(s.quantity_sold)
from sh.sales s, sh.customers c
where s.cust_id = c.cust_id and c.cust_id < 2
group by c.cust_id
SYS_SQL_PLAN_c17f5b75dc5f94e5  MANUAL-LOAD      YES YES NO  YES

SQL>

```

38. Determine the plan used for your statement when executing it. What do you observe?

The next time that you execute the query, it uses the plan baseline and the SQL profile. Use the `explain_query5.sql` script.

```

SQL> @explain_query5.sql
SQL> set echo on
SQL>
SQL> explain plan for
       2  select /*+ USE_NL(s c) FULL(s) FULL(c) */
       3         c.cust_id, sum(s.quantity_sold)
       4  from sh.sales s, sh.customers c
       5  where s.cust_id = c.cust_id and c.cust_id < 2
       6  group by c.cust_id;

```

**Practice 13-2: SQL Plan Management (SPM) (continued)**

```

Explained.

SQL>
SQL> select * from table(dbms_xplan.display(null,
2                                     null, 'basic +note'));

PLAN_TABLE_OUTPUT
-----
Plan hash value: 3070788227
-----
| Id | Operation                                | Name |
-----|-----|-----|
|  0 | SELECT STATEMENT                        |      |
|  1 |   HASH GROUP BY                        |      |
|  2 |     NESTED LOOPS                        |      |
|  3 |       PARTITION RANGE ALL              |      |
|  4 |         TABLE ACCESS BY LOCAL INDEX ROWID | SALES |
|  5 |           BITMAP CONVERSION TO ROWIDS   |      |

PLAN_TABLE_OUTPUT
-----
|  6 |           BITMAP INDEX RANGE SCAN       | SALES_CUST_BIX |
|  7 |           INDEX UNIQUE SCAN             | CUSTOMERS_PK   |
-----

Note
-----
- SQL profile "SPM_SQL_PROF" used for this statement

18 rows selected.

SQL>

```

39. Execute the `cleanup_spm.sql` script to purge your environment for this practice.

```

SQL> @cleanup_spm.sql
SQL> set echo on
SQL>
SQL> exec dbms_sqltune.drop_sql_profile(-
>      name => 'SPM_SQL_PROF');

PL/SQL procedure successfully completed.

SQL>
SQL> exec :cnt := dbms_spm.drop_sql_plan_baseline(-
>      'SYS_SQL_18a1ef14c17f5b75');

PL/SQL procedure successfully completed.

SQL>

```

### **Practice 13-2: SQL Plan Management (SPM) (continued)**

```
SQL> select signature, sql_handle, sql_text, plan_name,  
2         origin, enabled, accepted, fixed, autopurge  
3   from dba_sql_plan_baselines  
4  where sql_text like  
5         'select /*+ USE_NL(s c) FULL(s) FULL(c) */%';  
  
no rows selected  
  
SQL>  
SQL> select signature, category, name, sql_text  
2   from dba_sql_profiles where name like 'SPM%';  
  
no rows selected  
  
SQL> alter system set optimizer_use_sql_plan_baselines=FALSE;  
  
System altered.  
  
SQL> exit
```

40. Prepare for the next practice. Use the `prepare` command with the `CAPTURE` option. This script is in the `$HOME/workshops` directory. Exit EM before running the script. This script takes about 9 minutes.

```
$ cd $HOME/workshops  
$ ./prepare CAPTURE
```

## Practices for Lesson 14

In this practice, you use Database Replay.

## Practice 14-1: Using Database Replay

In this practice, you capture a workload, process the workload, and replay the workload with changes.

1. If you have not already done so, execute the `prepare` command with the `CAPTURE` option. This script performs a flashback database, restoring the database to a known state, and starts the database with the parameters set to simulate a 10g database with manual SGA tuning. This script is in the `$HOME/workshops` directory. Exit EM before running the script.


```
$ cd $HOME/workshops
$ ./prepare CAPTURE
```

2. Create a directory to capture the workload files named `$ORACLE_BASE/capture`.

```
$ mkdir $ORACLE_BASE/capture
```

3. Use Enterprise Manager to start a workload capture. Start EM and log in.

**Note:** Screenshots of some of the steps described in the table are provided, as noted in the Page column, below the table. Refer to these screenshots as necessary.

Step	Page	Action		
a	Database home	Click the Software and Support tab.		
b	Software and Support	Click Database Replay.		
c	Database Replay	Click the Go to Task (  ) icon to the right of the Capture Workload task.		
d	Capture Workload: Plan Environment	Acknowledge the prerequisites. (Select the check boxes.)		
		Click Next.		
e	Capture Workload: Options	Select “Do not restart the database prior to the capture.” Normally, you would restart the database to be sure to capture all transactions and not partial transactions.		
		Review the filter mode and exclusions. Do not change the exclusions.		
		Click Next.		
f	Workload Capture:	Set values.	Field	Value

### Practice 14-1: Using Database Replay (continued)

	Parameters		Capture Name	CAPTURE_10gSim
		Click Create Directory Object.		
g	Create Directory Object	Set values.	<b>Field</b>	<b>Value</b>
			Name	CAPTURE_DIR
			Path	/u01/app/oracle/capture
		Click Test File System.		
h	Host Login	Enter the host credentials: Username: oracle Password: oracle		
		Click Login.		
i	Confirmation	Click Return.		
j	Create Directory Object	Click OK.		
k	Workload Capture: Parameters	Click Next.		
l	Workload Capture: Schedule (See the example screenshot, which follows the table.)	Set values.	<b>Field</b>	<b>Value</b>
		Set host credentials.	Username	oracle
			Password	oracle
			Confirm password	oracle



**Practice 14-1: Using Database Replay (continued)**

		Click Next.
m	Capture Workload: Review	Click Submit.
n	Confirmation	Click Yes.
o	View Workload Capture: CAPTURE_10gSim	

# Practice 14-1: Using Database Replay (continued)

l) The following screenshot displays the Capture Workload: Schedule page:

**Capture Workload: Schedule**

Database **orcl.us.oracle.com**
Cancel
Back
Step 4 of 5
Next

Logged In As **SYS**

---

**Job Parameters**

Job Name 
Description

---

**Job Schedule**

Choose a start time and a capture duration so that the workload you are interested in replaying at a later time can be captured.

**Start**

☒ Immediately
☐ Later

Date 
Time 
AM PM

**Capture Duration**

☒ Not Specified
☐ Duration

Hours  Minutes

---

**Job Credentials**

**Host Credentials**

Username 
Password 
Confirm Password

☐ Save as Preferred Credential

- Start the charbench workload. Press the **Enter** key according to the instructions to start the workload. After about five minutes, stop the workload by pressing Enter.

```

$ cd /home/oracle/swingbench/bin
$ ./charbench
Author   :      Dominic Giles
Version  :      2.2

Results will be written to results.xml.
Hit Return to Start & Terminate Run...

Users: 30   TPM: 417   Nested TPM: 0
Users:  0   TPM: 343   Nested TPM: 0
Completed Run.

```

- Stop the capture and export the AWR data.

Step	Page	Action
a	View Workload Capture: CAPTURE-10gSIM	Click Stop Capture.
b	Confirmation	Click Yes.
c	Export AWR Data	Click Yes.

## Practice 14-1: Using Database Replay (continued)

d	View Workload Capture: CAPTURE_10gsim	Click View Job.
e	Scheduler Jobs	Click Refresh until the job no longer appears in the running jobs pane. This may take 3 minutes.

6. Process the captured workload.

Step	Page	Action		
a	View Workload Capture: CAPTURE-10gSIM	Click the Database tab.		
b	Database home	Click the Software and Support tab.		
c	Software and Support	Click Database Replay in the Real Application Testing section.		
d	Database Replay	Click the Go to Task icon to the right of the Preprocess Captured Workload task.		
e	Preprocess Captured Workload	Set Directory Object to CAPTURE_DIR.		
		Click Preprocess Workload.		
f	Preprocess Captured Workload: Database Version	Click Next.		
g	Preprocess Captured Workload: Schedule	Set values.	<b>Field</b>	<b>Value</b>
		Set host credentials.	Username	oracle
			Password	oracle
			Confirm password	oracle
		Click Next.		
h	Preprocess Captured Workload: Review	Click Submit.		
i	Database Replay	In the Confirmation message, click View Job.		

## Practice 14-1: Using Database Replay (continued)

j	Execution: orcl.us.oracle.com	Refresh until the status shows succeeded. This takes about 45 seconds.
---	----------------------------------	--

- Adjust the database setup. Execute the flashback REPLAY script. This script restores the database to the same point (SCN, where the capture started) and changes the parameters to use 11g optimizer and Automatic Memory Management. Log out of EM before you run `./prepare REPLAY`.

```
$ cd $HOME/workshops
$ ./prepare REPLAY
```

- Log in to EM and replay the workload.

Step	Page	Action		
a	Current page	Click the Database tab.		
b	Database home	Click the Software and Support tab.		
c	Software and Support	Click Database Replay in the Real Application Testing section.		
d	Database Replay	Click the Go to Task icon to the right of the Replay Workload task.		
e	Replay Workload	Click Create Directory Object.		
f	Create Directory Object	Set values.	Field	Value
			Name	CAPTURE_DIR
			Path	/u01/app/oracle/capture
		Click Test File System.		
g	Host Login	Enter host credentials: Username: oracle Password: oracle		
		Click Login.		
h	Confirmation	Click Return.		
i	Create Directory Object	Click OK.		

## Practice 14-1: Using Database Replay (continued)

j	Replay Workload	Click Setup Replay.
k	Replay Workload: Prerequisites	Notice the advice given for setup. The database has been restored, the database environment has been changed, and there are no external references in this database. The only thing left to do is to set up the replay clients.
		Click Continue.
l	Replay Workload: References to External Systems	Click Continue.
m	Replay Workload: Choose Initial Options	Click Next.
n	Replay Workload: Customize Options	Click Next.
o	Replay Workload: Prepare Replay Clients	Open a terminal window and run the workload replay client in calibrate mode.
		<code>\$ cd /u01/app/oracle/capture</code> <code>\$ wrc mode=calibrate</code>
		Type in the command to start the workload replay client in replay mode, but <i>do not start it</i> .
		<code>\$ wrc system/oracle_4U@orcl mode=replay</code>
		In EM, Click Next.
p	Replay Workload: Wait for Client Connections	Switch to the terminal window and start the workload client.
		When the client connection appears on the connection list, click Next.

### Practice 14-1: Using Database Replay (continued)

q	Replay Workload: Review	Click Submit.
---	----------------------------	---------------

As the replay progresses, the chart changes when the page is refreshed.

**Note:** The first update may take a few minutes.

When the replay completes, observe Divergence. The Divergence section should show zeros for Error Divergence and Data Divergence.

Expand the Detailed Comparison: Examine Detailed Comparison section. It shows the differences in several areas. The replay may be faster or slower. This section does not show whether the workload is actually performing better or worse.

9. Create a Compare Periods Report.

Step	Page	Action
a	View Replay Workload	Click the Report tab after the workload is completed. (The Report tab is not available while the replay is running.)
b	View Replay Workload Report subpage	Click Run AWR Compare Period Report in the Compare Period section.
c	Import AWR Data	Click Yes.
d	Select Schema	Enter SYSTEM.
		Click Import AWR Data.
e	View Replay Workload, Report tab	Click View Job.
f	Scheduler Jobs	Click Refresh until the Import job no longer appears on the Running tab. This job takes about a minute.
		Click the Back button on the browser.
		Click the Reload button on the browser.
g	View Replay Workload, Report tab	In the AWR Report section, test that you can produce an AWR report from the Capture data.  Set the Workload Capture or Replay field to Capture-10gSIM.  Click Run Report.

**Practice 14-1: Using Database Replay (continued)**

h	WORKLOAD REPOSITORY report for	If the report appears, the import was successful. Close the report window.
i	View Replay Workload, Report tab	Click Run AWR Compare Period Report in the Compare Period Report section.

10. When the report appears, review the report.

- a) Notice the difference between Elapsed time and DB Time.

	Elapsed Time	DB Time
<b>1<sup>st</sup> Period</b>		
<b>2<sup>nd</sup> Period</b>		

- b) Are there some areas where the performance seems to be worse?  
Check Top Timed Events and Time Model. (Look for places where % of DB time increased.)
- c) What is the overall performance difference?  
(In Time Model, DB Time in seconds is much reduced.  
**Note:** The DB CPU in % DB Time is increased because the total DB Time is much smaller.)
- d) What can you determine about individual SQL statement performance?  
(Statistics on all SQL statements are not reported. In each of the categories—Elapsed Time, CPU Time, Buffer Gets, and others—only the top 10 statements are reported. You could get some comparisons, but DB Replay is intended for overall performance comparison. SQL Performance Analyzer is intended for comparison of an entire set of SQL statements.)

11. From the \$HOME/workshops directory, use the `./cleanup REPLAY` command to reset the database environment for the next practice.

```
$ cd $HOME/workshops
$ ./cleanup REPLAY
```

12. Perform the first two steps for Practice 15-1, and then stop.

## Practices for Lesson 15

In this practice, you tune the shared pool.



## Practice 15-1: Sizing the Shared Pool

In this practice, you use the ADDM reports to diagnose and fix the shared pool size.

- 1) Prepare the database instance for this scenario.
  - a) Log out of Enterprise Manager.
  - b) In a terminal window, run the `./prepare SP` command.

```
$ cd /home/oracle/workshops
$ ./prepare SP
Database closed.
Database dismounted.
ORACLE instance shut down.
ORACLE instance started.

Total System Global Area 171581440 bytes
Fixed Size                 1335248 bytes
Variable Size              134217776 bytes
Database Buffers           33554432 bytes
Redo Buffers                2473984 bytes
Database mounted.
Database opened.
Finished Shared Pool setup
$
```

- 2) In a terminal window, determine the machine time and run the workload generator. The `workgen` script creates an AWR snapshot and a Statspack snapshot, and then runs the workload. Write the time returned by the `date` command: \_\_\_\_\_
 

```
date
./workgen SP
```

```
$ date
Tue Feb 23 12:24:25 UTC 2010
$ ./workgen SP

PL/SQL procedure successfully completed.

PL/SQL procedure successfully completed.
$
```

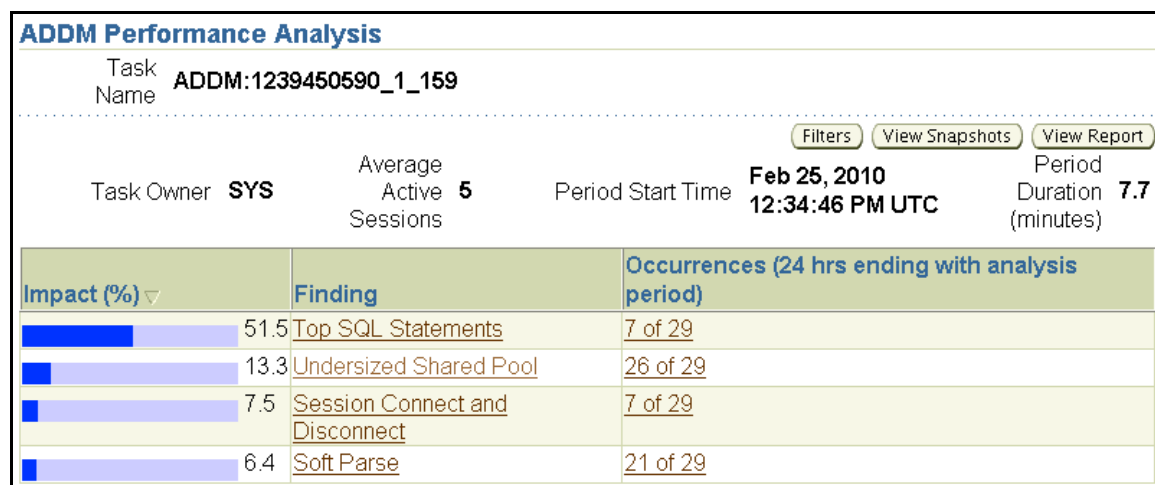
- 3) Check the date-time with the `date` command; if it has been more than 15 minutes since the time you recorded in step 2, continue, else wait until 15 minutes have elapsed. In a terminal window, stop the workload with the `rm runload` command.

```
$ rm runload
```

### Practice 15-1: Sizing the Shared Pool (continued)

- 4) Log in to Enterprise Manager. Click the Performance tab to see the Performance page. Wait until the Average Active Sessions graph has dropped to almost 0. Create an ADDM report. (Click Run ADDM Now, and then click Yes on the Confirmation page.) Notice any findings referring to the shared pool. These finding may include Hard Parse, Soft Parse, and Undersized Shared Pool.

The particular findings and the order of the findings may vary from the solution shown.



- 5) Create an AWR report between the last two snapshots. On the ADDM Report page, click View Snapshots to navigate to the snapshots page. Click the Report tab to generate the AWR report.
- 6) View the AWR report and note the values for the Top 5 Timed Foreground Events, Time Model Statistics, Instance Efficiencies, and the Library Cache Activity Statistics for the SQL AREA namespace.
- a) The Top 5 Timed Foreground Events may show time waiting on the library cache mutex, library cache load lock, shared pool latch, cursor pin waits, or row cache objects. Any of these indicate a possible problem in the shared pool.

Top 5 Timed Foreground Events					
Event	Waits	Time(s)	Avg wait (ms)	% DB time	Wait Class
DB CPU		249		10.86	
cursor: pin S wait on X	172	82	475	3.57	Concurrency
latch: shared pool	77	49	633	2.13	Concurrency
library cache load lock	59	45	755	1.94	Concurrency
library cache: mutex X	12	34	2875	1.51	Concurrency

- b) Time Model Statistics, in this example, shows a significant % of DB Time being used for hard and soft parses (parse time elapsed).

## Practice 15-1: Sizing the Shared Pool (continued)

### Time Model Statistics

Statistic Name	Time (s)	% of DB Time
sql execute elapsed time	1,694.03	73.99
parse time elapsed	588.84	25.72
hard parse elapsed time	441.95	19.30
DB CPU	248.58	10.86
connection management call elapsed time	171.73	7.50
hard parse (sharing criteria) elapsed time	43.52	1.90
hard parse (bind mismatch) elapsed time	30.16	1.32
failed parse elapsed time	11.90	0.52
PL/SQL execution elapsed time	10.02	0.44
PL/SQL compilation elapsed time	9.09	0.40
repeated bind elapsed time	0.72	0.03
sequence load elapsed time	0.51	0.02
DB time	2,289.62	
background elapsed time	25.27	
background cpu time	2.42	

- c) The latch and mutex waits taken with the hard parses shown in Time Model leads to a check of the Load Profile section. In this example, hard parses are a significant percentage of all parses.

**Practice 15-1: Sizing the Shared Pool (continued)**

Load Profile				
	Per Second	Per Transaction	Per Exec	Per Call
DB Time(s):	5.0	12.6	0.01	0.01
DB CPU(s):	0.5	1.4	0.00	0.00
Redo size:	11,942.5	30,309.5		
Logical reads:	5,176.9	13,138.7		
Block changes:	55.7	141.3		
Physical reads:	1,647.2	4,180.4		
Physical writes:	3.9	9.8		
User calls:	732.7	1,859.7		
Parses:	288.0	731.0		
Hard parses:	61.1	155.1		
W/A MB processed:	0.9	2.3		
Logons:	0.7	1.8		
Executes:	591.5	1,501.3		
Rollbacks:	0.0	0.0		
Transactions:	0.4			

- d) Navigate to Library Cache Statistics from the Main menu. Check the Reloads statistic of the SQL AREA namespace in the Library Cache Activity Statistics section. The hard parses can be caused by SQL that cannot be shared, or by cursors aging out of the shared pool before they can be reused. This example shows a high value for reloads and a high Pct Miss for Gets and Pins. This indicates that the cursor could have been shared, but the shared pool is too small. Reloads should be less than 1% of Pin Requests.

**Practice 15-1: Sizing the Shared Pool (continued)****Library Cache Activity**

Namespace	Get Requests	Pct Miss	Pin Requests	Pct Miss	Reloads	Invali- dations
APP CONTEXT	1,764	1.70	1,764	1.70	0	0
BODY	1,127	9.41	3,571	10.78	267	0
CLUSTER	3,461	1.18	1,995	2.06	0	0
DBLINK	593	13.66	0		0	0
EDITION	304	27.63	606	27.89	0	0
INDEX	624	51.60	624	52.08	1	0
OBJECT ID	1	100.00	0		0	0
QUEUE	948	2.00	1,063	5.55	34	0
SCHEMA	18,550	0.06	0		0	0
SQL AREA	189,895	56.89	390,715	16.35	20,167	2,505
SUBSCRIPTION	1	100.00	1	100.00	0	0
TABLE/PROCEDURE	149,200	5.61	181,280	11.63	8,927	0
TRIGGER	4,534	1.87	5,204	2.21	16	0

e) Further investigation of the dictionary cache shows a high number of misses.

**Dictionary Cache Stats**

Cache	Get Requests	Pct Miss	Scan Reqs	Pct Miss	Mod Reqs	Final Usage
dc_awr_control	12	66.67	0		2	1
dc_files	17	100.00	0		0	0
dc_global_oids	11,372	6.79	0		0	9
dc_histogram_data	81,072	23.14	0		0	443
dc_histogram_defs	159,785	27.07	0		2	924
dc_object_grants	23,127	6.03	0		0	10
dc_objects	346,928	5.04	0		3	587
dc_profiles	326	13.50	0		0	0
dc_rollback_segments	65	0.00	0		0	22
dc_segments	117,677	13.73	0		7	191
dc_sequences	96	100.00	0		96	0
dc_tablespace	81,028	0.06	0		0	4
dc_users	170,746	0.66	0		0	64
global database name	1,022	2.25	0		0	1
kqlsubheap_object	155	48.39	0		0	0

- Find an optimum shared pool size for this workload. The solution can be obtained either through the EM tools or by using the Shared Pool Advisory in the Statspack report.

## Practice 15-1: Sizing the Shared Pool (continued)

- a) Navigate to the Memory Advisors page under the Server tab.

**Memory Advisors**

Page Refreshed **March 30, 2010 9:13:47 AM UTC** [Refresh](#)

[Show SQL](#) [Revert](#) [Apply](#)

When Automatic Memory Management is enabled, the database will automatically set the optimal distribution of memory. The distribution of memory will change from time to time to accommodate changes in the workload.

Automatic Memory Management **Disabled** [Enable](#)

---

**SGA** [PGA](#)

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.

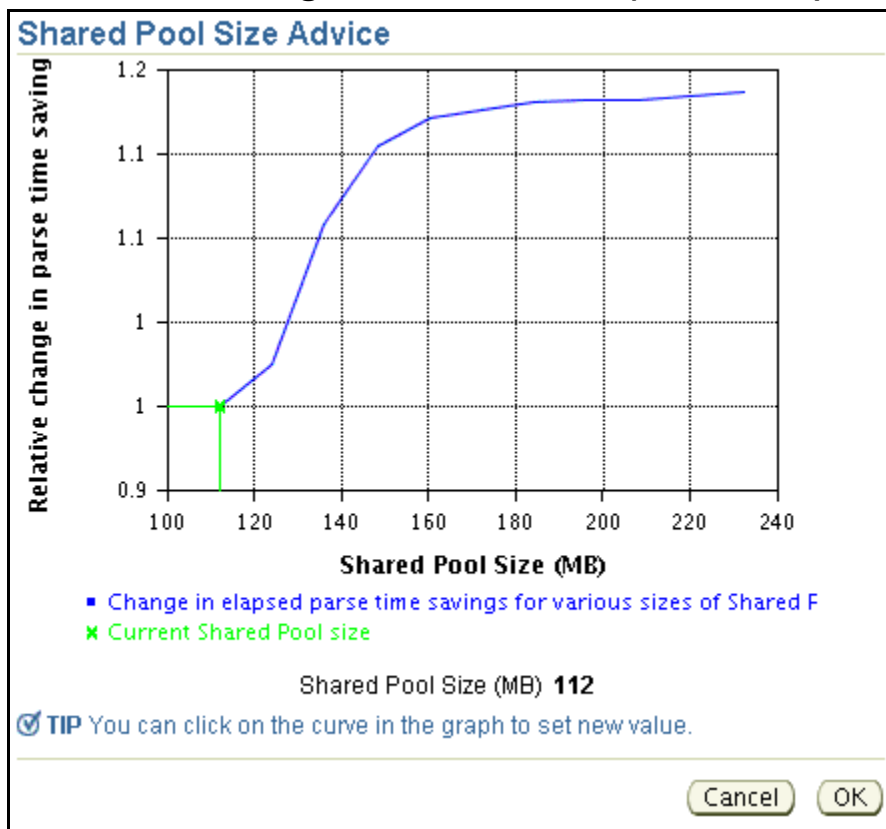
Automatic Shared Memory Management **Disabled** [Enable](#)

Shared Pool	<input type="text" value="112"/>	MB	<a href="#">Advice</a>
Buffer Cache	<input type="text" value="32"/>	MB	<a href="#">Advice</a>
Large Pool	<input type="text" value="4"/>	MB	
Java Pool	<input type="text" value="12"/>	MB	
Other (MB)	<input type="text" value="3"/>		
Total SGA (MB)	<b>163</b>		<a href="#">Calculate</a>

Memory Component	Percentage
Shared Pool	68.4%
Buffer Cache	19.6%
Large Pool	2.4%
Java Pool	7.3%
Other	2.2%

- b) Click Shared Pool Advice. The graph shows performance versus shared pool size. The point where the curve flattens is where there is no more improvement in performance as the shared pool size is increased. (The graph generated for your instance may vary.) Make a note of this point for your instance. Click Cancel.

**Note:** This graph displays estimates based on statistics collected during the previous workload. If the workload changes, so do the estimates.

**Practice 15-1: Sizing the Shared Pool (continued)**

- c) On the Memory Advisors Page, change the Shared Pool value to 200 MB. Click Calculate.

What is the calculated value? \_\_\_\_\_

If the calculated value (Total SGA) is **less than** the Maximum SGA size, click Apply.

If the calculated value (Total SGA) is **greater than** the Maximum SGA size, an “Insufficient memory to grow pool” error message appears.

Change the Maximum SGA size to the calculated value + 2 MB. Select the “Apply changes to SPFILE only” check box at the bottom of the page. Click Apply.

8. Run the `./prepare SP2` script from the `$HOME/workshops` directory. This script changes the `SHARED_POOL_SIZE` parameter to 200 MB; because, you are not using Automatic SGA Memory Management, the change requires the database to be restarted.

```
$ ./prepare SP2
Database closed.
Database dismounted.
ORACLE instance shut down.
ORACLE instance started.

Total System Global Area  263651328 bytes
Fixed Size                  1335864 bytes
Variable Size              226495944 bytes
```

**Practice 15-1: Sizing the Shared Pool (continued)**

```
Database Buffers          33554432 bytes
Redo Buffers              2265088 bytes
Database mounted.
Database opened.
Finished Shared Pool second setup
$
```

9. Record the current time: \_\_\_\_\_  
Run the workload generator again (. /workgen SP) for five minutes, and then stop the workload with the `rm runload` command.

a) Find the system time. Start the workload.

```
$ date
Thu Dec  1 08:10:03 PST 2005
$ ./workgen SP

PL/SQL procedure successfully completed.

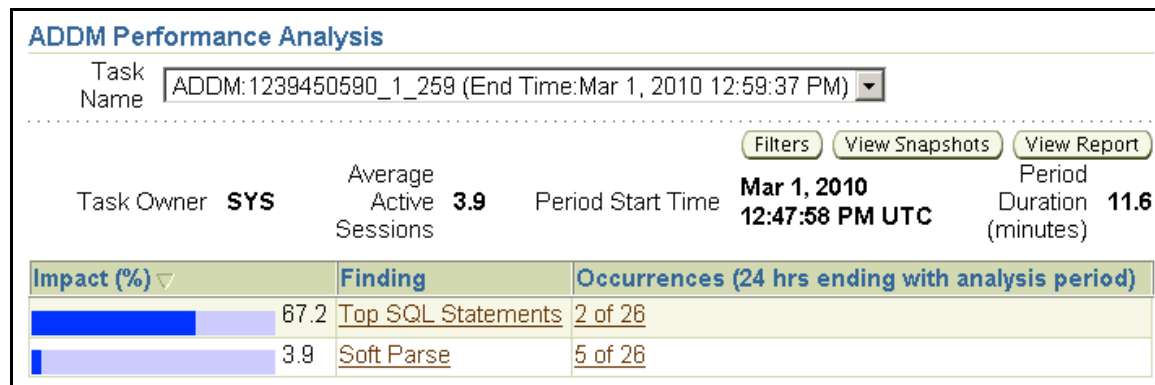
PL/SQL procedure successfully completed.
```

b) After five minutes, stop the workload.

```
$ rm runload
```

10. Log out of EM, log in to EM, navigate to the Performance page, and create another ADDM snapshot. What is the difference? What are the primary issues now?

a) Notice the findings. The hard-parse findings have been reduced or eliminated from the findings. There may be no findings.



11. From the ADDM page, click View Snapshots to navigate to the Snapshot Details page, and then click the Report tab to create an AWR report.



### Practice 15-1: Sizing the Shared Pool (continued)

12. View the AWR report and compare the values from the previous report.

**Note:** The actual values that you see may vary from those shown in the solutions.

- a) In the Top 5 Foreground Timed Events, the latch and mutex waits for the shared pool and cursors are reduced or eliminated.

Top 5 Timed Foreground Events					
Event	Waits	Time(s)	Avg wait (ms)	% DB time	Wait Class
DB CPU		328		12.17	
log file sync	350	23	65	0.85	Commit
kfk: async disk IO	202,150	12	0	0.45	System I/O
SQL*Net message to client	644,068	10	0	0.36	Network
latch free	18	9	478	0.32	Other

- b) Time Model (in the example) shows that hard parses are still contributing to parse elapsed time, but take a very small % of DB Time. Your values may vary.

**Practice 15-1: Sizing the Shared Pool (continued)****Time Model Statistics**

Statistic Name	Time (s)	% of DB Time
sql execute elapsed time	2,250.07	83.38
DB CPU	328.40	12.17
parse time elapsed	126.85	4.70
hard parse elapsed time	21.24	0.79
failed parse elapsed time	13.79	0.51
connection management call elapsed time	9.95	0.37
PL/SQL compilation elapsed time	2.64	0.10
PL/SQL execution elapsed time	1.47	0.05
repeated bind elapsed time	0.08	0.00
hard parse (sharing criteria) elapsed time	0.04	0.00
sequence load elapsed time	0.00	0.00
DB time	2,698.62	
background elapsed time	10.48	
background cpu time	0.68	

- c) The Load Profile section shows that hard parses are reduced (as are all parses).

Load Profile				
	Per Second	Per Transaction	Per Exec	Per Call
DB Time(s):	3.9	15.6	0.02	0.00
DB CPU(s):	0.5	1.9	0.00	0.00
Redo size:	5,126.2	20,692.2		
Logical reads:	4,098.1	16,542.2		
Block changes:	20.9	84.2		
Physical reads:	2,251.5	9,088.4		
Physical writes:	2.2	8.7		
User calls:	1,065.0	4,299.0		
Parses:	176.9	714.1		
Hard parses:	9.5	38.3		
W/A MB processed:	0.4	1.7		
Logons:	1.0	3.8		
Executes:	187.2	755.7		
Rollbacks:	0.0	0.0		
Transactions:	0.3			

## Practice 15-1: Sizing the Shared Pool (continued)

- d) In the Library Cache Activity section, reloads for the SQL AREA namespace are now close to an acceptable range. The % of Reloads versus Pin Requests is less than 4% in the example.

### Library Cache Activity

Namespace	Get Requests	Pct Miss	Pin Requests	Pct Miss	Reloads	Invali- dations
APP CONTEXT	259	0.00	259	0.00	0	0
BODY	2,112	2.04	3,906	1.28	0	0
CLUSTER	145	0.00	74	0.00	0	0
DBLINK	1,246	0.00	0		0	0
EDITION	637	0.00	1,271	0.00	0	0
INDEX	59	30.51	59	30.51	0	0
OBJECT ID	1	100.00	0		0	0
QUEUE	1,580	0.06	1,907	0.10	0	0
SCHEMA	3,213	0.25	0		0	0
SQL AREA	187,872	47.49	160,476	8.88	5,433	5,464
SUBSCRIPTION	1	100.00	1	100.00	0	0
TABLE/PROCEDURE	22,339	4.27	50,937	4.63	2	0
TRIGGER	5,632	0.05	5,654	0.05	0	0

- e) The Dictionary Cache Stats section shows a much reduced percentage of misses.

## Practice 15-1: Sizing the Shared Pool (continued)

### Dictionary Cache Stats

Cache	Get Requests	Pct Miss	Scan Reqs	Pct Miss	Mod Reqs	Final Usage
dc_awr_control	16	0.00	0		2	1
dc_global_oids	15,587	0.29	0		0	80
dc_histogram_data	11,056	9.35	0		0	2,126
dc_histogram_defs	15,952	20.46	0		2	5,374
dc_object_grants	26,506	0.29	0		0	109
dc_objects	53,595	2.39	0		3	3,188
dc_profiles	649	0.00	0		0	2
dc_rollback_segments	152	0.00	0		0	22
dc_segments	6,825	12.19	0		8	1,357
dc_sequences	3	100.00	0		3	3
dc_tablespace_quotas	4	50.00	0		0	2
dc_tablespaces	30,048	0.00	0		0	19
dc_users	81,144	0.02	0		0	97
global database name	1,101	0.00	0		0	1
kqlsubheap_object	11	18.18	0		0	2
outstanding_alerts	16	0.00	0		0	17

## Practice 15-2: Tuning a Hard-Parse Workload

In this section, you run a hard-parse workload, view the characteristic report events, and fix the problem.

1. Prepare the database for this scenario.
  - a) Log out of Enterprise Manager.
  - b) Set up the scenario by running the `./prepare HARDPARSE` script. Change the directory to the `/home/oracle/workshops` directory. Run the prepare script.

```
$ cd /home/oracle/workshops
$ ./prepare HARDPARSE
Database closed.
Database dismounted.
ORACLE instance shut down.

ORACLE instance started.

Total System Global Area 179949568 bytes
Fixed Size                 1335304 bytes
Variable Size             142610424 bytes
Database Buffers          33554432 bytes
Redo Buffers              2449408 bytes
Database mounted.
Database opened.
Finished setup Hard Parse
$
```

2. Run the workload generator: `./workgen HARDPARSE`.  
The workload generator creates an AWR snapshot and a Statspack snapshot at the beginning. An additional Statspack snapshot is taken at the end.
  - a) Run the workload generator.

```
$ ./workgen HARDPARSE

PL/SQL procedure successfully completed.

PL/SQL procedure successfully completed.

System altered.

PL/SQL procedure successfully completed.

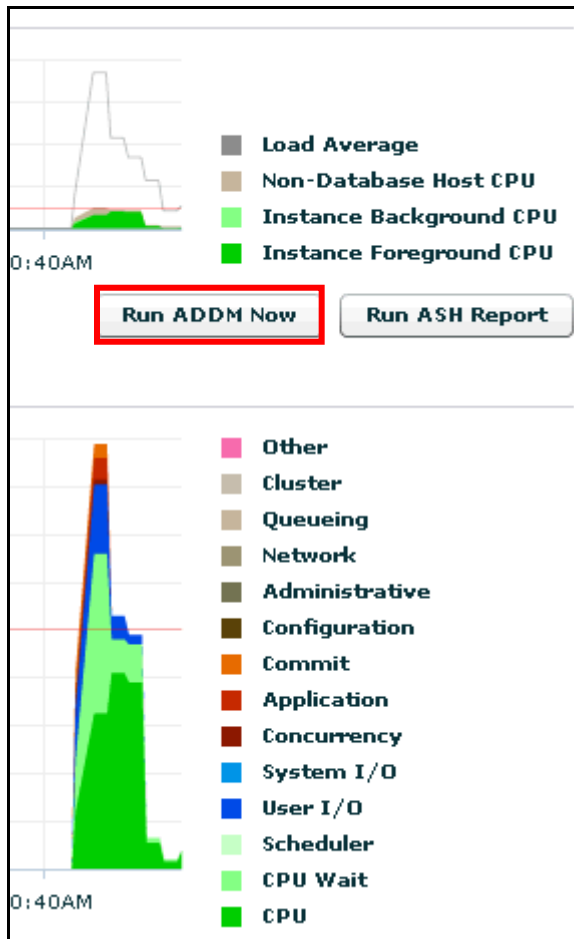
PL/SQL procedure successfully completed.
```

## Practice 15-2: Tuning a Hard-Parse Workload (continued)

PL/SQL procedure successfully completed.

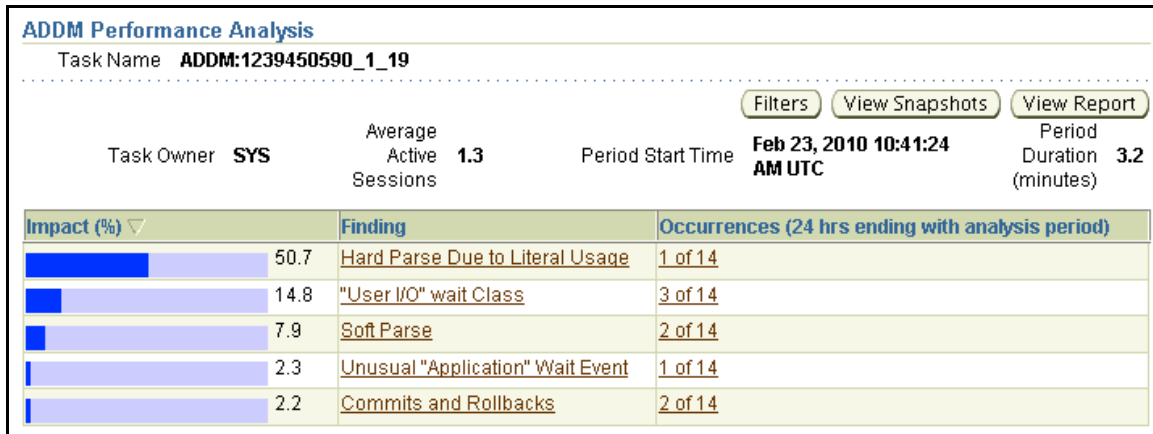
\$

- b) Log in to Enterprise Manager, navigate to the Database Performance page, wait until the workload begins to drop (about 5 minutes), and then click Run ADDM Now. Click Yes on the Confirmation page.



- c) View the ADDM report. Note that Benefit % is the same as % DB Time shown in the Top 5 Foreground Waits section of the AWR report. What is the value of Average Active Sessions? \_\_\_\_\_

## Practice 15-2: Tuning a Hard-Parse Workload (continued)



## Practice 15-2: Tuning a Hard-Parse Workload (continued)

- Drill down to the recommendations. Click the “Hard Parse Due to Literal Usage” finding.

**Recommendations**  
[Show All Details](#) | [Hide All Details](#)  

Details	Category	Benefit (%)
▼ <a href="#">Hide</a>	Application Analysis	<div><div></div></div> 50.7

Action **Alternatively, you may set the parameter "cursor\_sharing" to "force".** [Implement](#) [Filters](#)

Action **Investigate application logic for possible use of bind variables instead of literals.**

Rationale **At least 7 SQL statements with FORCE\_MATCHING\_SIGNATURE 14812360487617340174 and PLAN\_HASH\_VALUE 4238351645 were found to be using literals. Look in V\$SQL for examples of such SQL statements.**

**Findings Path**  
[Expand All](#) | [Collapse All](#)  

Findings	Percentage of Finding's Impact (%)	Additional Information
▼ SQL statements were not shared due to the usage of literals. This resulted in additional hard parses which were consuming significant database time.	<div><div></div></div> 50.7	
Hard parsing of SQL statements was consuming significant database time.	<div><div></div></div> 55.1	

- Based on the ADDM report, what is the recommended action?

The recommendation is to “set the CURSOR\_SHARING parameter to FORCE.”

- Create an AWR or Statspack report from the last two snapshots. The following shows the solution using AWR and Enterprise Manager.

Step	Page	Action
a	Current page	Navigate to the Snapshots page.
b	Snapshots	Select the next to last snapshot.
		Select View Report from Actions.
		Click Go.
c	View Report	Select the last snapshot.
		Click OK.

- Review the AWR report for the characteristics of a hard-parse problem. The goal is to reduce hard parses because the cost of a hard parse is at least an order of magnitude greater than the cost of a soft parse.
  - The Top 5 Timed Events show that there are no significant wait events. CPU Time by itself does not indicate a problem.



## Practice 15-2: Tuning a Hard-Parse Workload (continued)

### Top 5 Timed Foreground Events

Event	Waits	Time(s)	Avg wait (ms)	% DB time	Wait Class
DB CPU		137		54.97	
db file sequential read	14,746	35	2	14.03	User I/O
SQL*Net break/reset to client	1,862	6	3	2.26	Application
log file sync	118	5	46	2.18	Commit
buffer busy waits	2	3	1321	1.06	Concurrency

- b) Time Model gives the first indication of a problem. Parse time elapsed is a significant portion of DB Time, and hard parse elapsed time is the majority of the parse time elapsed.

### Time Model Statistics

- Total time in database user-calls (DB Time): 248.6s
- Statistics including the word "background" measure background process time, and so do not contribute to the DB time statistic
- Ordered by % or DB time desc, Statistic name

Statistic Name	Time (s)	% of DB Time
sql execute elapsed time	219.80	88.41
parse time elapsed	156.79	63.07
hard parse elapsed time	137.06	55.13
DB CPU	136.66	54.97
PL/SQL execution elapsed time	11.83	4.76
PL/SQL compilation elapsed time	4.56	1.84
hard parse (sharing criteria) elapsed time	3.00	1.21
connection management call elapsed time	1.88	0.75
hard parse (bind mismatch) elapsed time	0.94	0.38
repeated bind elapsed time	0.09	0.04
sequence load elapsed time	0.06	0.03
DB time	248.62	
background elapsed time	18.59	
background cpu time	1.13	

- c) In the Load Profile section, compare Parses per second to Hard parses per second. A majority of the parses are hard parses.

**Practice 15-2: Tuning a Hard-Parse Workload (continued)****Load Profile**

	Per Second	Per Transaction	Per Exec	Per Call
DB Time(s):	1.3	1.7	0.00	0.04
DB CPU(s):	0.7	1.0	0.00	0.02
Redo size:	30,586.1	40,724.9		
Logical reads:	7,741.5	10,307.7		
Block changes:	128.5	171.1		
Physical reads:	97.8	130.3		
Physical writes:	16.4	21.9		
User calls:	32.2	42.9		
Parses:	448.1	596.6		
Hard parses:	343.2	456.9		
W/A MB processed:	1.5	2.1		
Logons:	0.3	0.3		
Executes:	586.8	781.3		
Rollbacks:	0.0	0.0		
Transactions:	0.8			

- d) Further investigation into the Instance Efficiencies Percentages section shows that % Non-Parse CPU is low, indicating that much of the CPU time is spent parsing. The Soft Parse %: shows that very few statements are found in the cache.

**Instance Efficiency Percentages (Target 100%)**

Buffer Nowait %:	100.00	Redo NoWait %:	100.00
Buffer Hit %:	98.82	In-memory Sort %:	100.00
Library Hit %:	70.64	Soft Parse %:	23.42
Execute to Parse %:	23.63	Latch Hit %:	100.00
Parse CPU to Parse Elapsed %:	68.37	% Non-Parse CPU:	31.94

7. Implement the ADDM recommendation: Either click the Implement button on the ADDM report or use the `ALTER SYSTEM SET CURSOR_SHARING = FORCE SCOPE=BOTH` command.
  - a) Using the SQL\*Plus interface, execute the `ALTER SYSTEM SET CURSOR_SHARING = FORCE SCOPE=BOTH` command.

## Practice 15-2: Tuning a Hard-Parse Workload (continued)

```
$ sqlplus / as sysdba

...

SQL> ALTER SYSTEM SET CURSOR_SHARING = FORCE SCOPE=BOTH;

System altered.

SQL> exit
Disconnected from Oracle Database 10g Enterprise Edition
Release 10.2.0.1.0 - Production
With the Partitioning, OLAP and Data Mining options
$
```

8. Run the workload generator again and observe the differences in the tool of your choice.
  - a) Run the ./workgen HARDPARSE script.
  - b) After about 3 minutes, run ADDM from the Performance page as in step 2. The load graph will be much reduced.
  - c) Using the ADDM tool, you see the impact of changing the CURSOR\_SHARING parameter. The performance penalty of hard-parsing has dropped significantly. The hard-parse finding may not appear in the ADDM report. It is possible that there will be no findings. Notice that the value for Average Active Sessions has dropped significantly. This indicates that fewer sessions are waiting.

ADDM Performance Analysis				
Task Name <b>ADDM:1239450590_1_22</b>				
Task Owner <b>SYS</b>		Average Active Sessions <b>0.3</b>	Period Start Time <b>Feb 23, 2010 11:00:50 AM UTC</b>	Period Duration <b>3.2</b> (minutes)
<a href="#">Filters</a> <a href="#">View Snapshots</a> <a href="#">View Report</a>				
Impact (%) ▾	Finding	Occurrences (24 hrs ending with analysis period)		
19.8	<a href="#">Undersized Shared Pool</a>	2 of 17		
19.7	<a href="#">"User I/O" wait Class</a>	5 of 17		
9	<a href="#">Hard Parse Due to Sharing Criteria</a>	3 of 17		
7.1	<a href="#">Soft Parse</a>	3 of 17		
4.7	<a href="#">PL/SQL Compilation</a>	2 of 17		
4.4	<a href="#">Commits and Rollbacks</a>	3 of 17		
2.3	<a href="#">Session Connect and Disconnect</a>	1 of 17		

- d) Create an AWR report. On the ADDM page, click View Snapshots. On the Snapshot detail page, click the Report tab.
- e) Using the AWR report, review the significant statistics. Notice that all significant statistics show improvement. Notice that hard parses are a much smaller percentage of the parses, Soft Parse % is much improved, and % Non-Parse CPU is much higher. You may have that noticed that the workload script ran more quickly as well.

### Practice 15-2: Tuning a Hard-Parse Workload (continued)

Load Profile				
	Per Second	Per Transaction	Per Exec	Per Call
DB Time(s):	0.3	1.1	0.00	0.04
DB CPU(s):	0.2	0.7	0.00	0.02
Redo size:	15,565.7	67,046.8		
Logical reads:	6,253.0	26,933.9		
Block changes:	38.2	164.3		
Physical reads:	135.1	581.8		
Physical writes:	10.5	45.1		
User calls:	6.4	27.5		
Parses:	365.5	1,574.5		
Hard parses:	7.2	31.1		
W/A MB processed:	0.6	2.4		
Logons:	0.1	0.4		
Executes:	441.0	1,899.7		
Rollbacks:	0.0	0.0		
Transactions:	0.2			

## Practice 15-2: Tuning a Hard-Parse Workload (continued)

### Time Model Statistics

- Total time in database user-calls (DB Time): 50.6s
- Statistics including the word "background" measure background process time, and so do not contribute to the DB time statistic
- Ordered by % of DB time desc, Statistic name

Statistic Name	Time (s)	% of DB Time
sql execute elapsed time	45.64	90.22
DB CPU	30.62	60.53
parse time elapsed	20.86	41.23
hard parse elapsed time	17.28	34.15
PL/SQL execution elapsed time	6.83	13.51
hard parse (sharing criteria) elapsed time	4.57	9.03
PL/SQL compilation elapsed time	2.35	4.65
connection management call elapsed time	1.15	2.27
hard parse (bind mismatch) elapsed time	0.35	0.69
repeated bind elapsed time	0.09	0.18
sequence load elapsed time	0.00	0.00
DB time	50.59	
background elapsed time	4.62	
background cpu time	0.31	

### Instance Efficiency Percentages (Target 100%)

Buffer Nowait %:	100.00	Redo NoWait %:	100.00
Buffer Hit %:	97.84	In-memory Sort %:	100.00
Library Hit %:	94.03	Soft Parse %:	98.02
Execute to Parse %:	17.12	Latch Hit %:	100.00
Parse CPU to Parse Elapsed %:	78.67	% Non-Parse CPU:	81.09

### Practice 15-3: Keeping Objects in the Shared Pool

The performance of some applications is hindered by having to reload frequently used objects into the shared pool. If an object (such as a procedure, package, or sequence) is used often but not frequently enough to prevent aging out of the shared pool, keeping it will reduce the number of reloads. Keeping objects that are used very frequently is not necessary, but it does not hurt performance because those objects will stay in the shared pool without being kept.

- 1) Start the workload generator: `./workgen KEEP`.
- 2) After the workload has run for a few minutes, choose a candidate object to KEEP in the shared pool. Write and save a query to find a candidate object. Write the query to test whether the object has been KEPT. A suggested query is:

```
SELECT name, type, sharable_mem, kept
FROM v$db_object_cache
WHERE sharable_mem > 4000
AND EXECUTIONS > 5
AND (type='FUNCTION' OR type='PROCEDURE')
/
```

This query is available in the `$HOME/workshops` directory as `sp_objects.sql`. The results of the query will vary. The `DEPLETE_INV` procedure should appear in all queries.

```
$ cd $HOME/workshops
$ sqlplus / as sysdba

SQL> @sp_objects.sql
SQL> set echo on
SQL>
SQL> set pagesize 100
SQL>
SQL> SELECT name, type, sharable_mem, kept
2      FROM v$db_object_cache
3      WHERE sharable_mem > 4000
4      AND EXECUTIONS > 5
5      AND (type='FUNCTION' OR type='PROCEDURE')
6  /
```

NAME

TYPE	SHARABLE_MEM	KEP
GETEMKEY		
FUNCTION	8576	NO
DEPLETE_INV		
PROCEDURE	16768	YES
SETEMUSERCONTEXT		
PROCEDURE	20880	NO

### Practice 15-3: Keeping Objects in the Shared Pool (continued)

```
DECRYPT
FUNCTION                                20864    NO

SQL>
```

3. Use the KEEP procedure to keep the DEPLETE\_INV PROCEDURE object in the shared pool.

```
SQL> EXECUTE dbms_shared_pool.keep('OE.DEPLETE_INV');
```

PL/SQL procedure successfully completed.

```
SQL> @sp_objects.sql
```

```
SQL> set echo on
```

```
SQL>
```

```
SQL> set pagesize 100
```

```
SQL>
```

```
SQL> SELECT name, type, sharable_mem, kept
2         FROM v$db_object_cache
3         WHERE sharable_mem > 4000
4         AND EXECUTIONS > 5
5         AND (type='FUNCTION' OR type='PROCEDURE')
6  /
```

NAME

NAME	TYPE	SHARABLE_MEM	KEP
GETEMKEY	FUNCTION	8576	NO
DEPLETE_INV	PROCEDURE	16768	YES
SETEMUSERCONTEXT	PROCEDURE	20880	NO
DECRYPT	FUNCTION	20864	NO

4. Flush the shared pool and observe the state of KEPT objects.

```
SQL> ALTER SYSTEM FLUSH SHARED_POOL;
```

System altered.

```
SQL> @sp_objects
```

**Practice 15-3: Keeping Objects in the Shared Pool (continued)**

```

SQL> set echo on
SQL>
SQL> set pagesize 100
SQL>
SQL> SELECT name, type, sharable_mem, kept
2      FROM v$db_object_cache
3      WHERE sharable_mem > 4000
4      AND EXECUTIONS > 5
5      AND (type='FUNCTION' OR type='PROCEDURE')
6  /

```

NAME

-----

TYPE SHARABLE\_MEM KEPT

-----

DEPLETE\_INV

PROCEDURE 16768 YES

SQL> **exit**

5. Stop the workload generator by executing the `rm runload` command.

```
$ rm runload
```

6. Clean up the database by running the `./cleanup SP` script.

```

$ ./cleanup SP
Cleanup Finished

```



## Practices for Lesson 16

The goal of this practice is to observe the symptoms of, diagnose, and apply solutions for buffer cache problems.

## Practice 16-1: Using the DB Cache Advisor

In this practice, assume that you have an OLTP workload that needs some additional tuning and that the SQL statements are not available to tune, or have already been tuned using the SQL Tuning Advisor and SQL Access Advisor. You use the diagnostics in the AWR report and the DB Cache Advisor to choose an optimal size for the database buffer cache. The workload generator for this scenario creates an AWR snapshot, and then runs the workload.

1. Complete the setup for this scenario.
  - a) Log out of the Enterprise Manager.
  - b) Run `./prepare BC`.

```
$ cd $HOME/workshops
$ ./prepare BC
Database closed.
Database dismounted.
ORACLE instance shut down.

ORACLE instance started.

Total System Global Area  284585984 bytes
Fixed Size                  1335980 bytes
Variable Size              268438868 bytes
Database Buffers           8388608 bytes
Redo Buffers                6422528 bytes
Database mounted.
Database opened.
Finished setup Buffer Cache
$
```

2. Verify that `DB_CACHE_SIZE` is set to 8M.

```
$ sqlplus / as sysdba

SQL> show parameter DB_CACHE_SIZE

NAME                                TYPE                                VALUE
-----                                -                                -
db_cache_size                       big integer                         8M

SQL> exit
$
```

3. Start the workload generator with `./workgen BC`. The `workgen` script creates an AWR snapshot and a Statspack snapshot before starting the workload in the background.

```
$ ./workgen BC

PL/SQL procedure successfully completed.
```

## Practice 16-1: Using the DB Cache Advisor (continued)

```
PL/SQL procedure successfully completed.
```

```
$
```

4. Wait for about five minutes, and then in a terminal window, stop the workload with the `rm runload` command.

```
$ rm runload
```

5. Log in to Enterprise Manager. From the Performance page, click Run ADDM Now to generate an ADDM report. View Performance Analysis and check the performance findings. Examine the top finding that you, as a DBA, can address.
  - a) Review ADDM Performance Analysis. When you select the findings with the greatest impact, you may find several that you can do nothing about, such as “Top SQL Statements” shown in the following example. Assume that you have already tuned these statements as covered in the previous lessons and move down the list of findings. Other findings that you may see are: “CPU Usage,” which usually recommends SQL tuning and “User I/O wait Class,” which requires more analysis, sometimes leading to a change in the buffer cache size. The finding of interest in the example list is “Undersized Buffer Cache.” The finding “Free Buffer waits” is often associated with an undersized buffer cache.

ADDM Performance Analysis				
Task Name		ADDM:1239450590_1_555		
Task Owner		SYS	Average Active Sessions	0.8
Period Start Time		Mar 10, 2010 7:26:26 AM UTC		Filters View Snapshots View Report
Period Duration		8 (minutes)		
Impact (%)	Finding	Occurrences (24 hrs ending with analysis period)		
51.2	Top SQL Statements	18 of 41		
18	Undersized Buffer Cache	17 of 41		
3.8	Hard Parse	17 of 41		

- b) Drill down on the “Undersized Buffer Cache” finding. Notice the recommendation to increase the buffer cache size to 12 MB. This value is limited by the advisor, which takes estimates only from 50% to 200% of the current setting.
6. Create an AWR report. Save the report as `awr_BC_1.html`.
  - a) On the ADDM report page, click View Snapshots. On the Snapshot Details page, click the Report tab.
  - b) Set the download preferences. Note that these instructions are for Firefox 3.0.15. In the Browser menu, click Edit > Preferences. In the Preferences dialog box, in the Downloads section, select “Always ask me where to save files.” Click Close.

### Practice 16-1: Using the DB Cache Advisor (continued)

- c) At the top right of the report, click “Save to File.” When prompted, specify the name of the report. Notice which directory the report is being created in so that you can find it later, and write the directory path here.
- 

7. Review the AWR report and identify the primary performance issue.
- a) The Top 5 Timed Foreground Events section shows one or more of the events that point to the buffer cache size: db file sequential read, db file scattered read, buffer busy waits, free buffer waits, latch: cache buffer lru chain, or latch: cache buffers chains.

From the report, determine the possible impact of the buffer cache problem. If all waits associated with the physical reads are eliminated, what percent increase in performance could you expect? \_\_\_\_\_ (10 to 25 % is expected. The following example shows ~17% = db file sequential reads + latch: cache buffers chains.) Of course, removing all the waits for physical reads is not possible. Reducing the physical reads will often reduce the DB CPU.

**Note:** Your results will vary.

Top 5 Timed Foreground Events					
Event	Waits	Time(s)	Avg wait (ms)	% DB time	Wait Class
DB CPU		257		64.05	
db file sequential read	3,722,448	67	0	16.69	User I/O
latch: cache buffers chains	59	3	51	0.76	Concurrency
direct path read	105,223	2	0	0.60	User I/O
direct path write temp	63	1	19	0.30	User I/O

- b) What is Time Model showing as the greatest loads? \_\_\_\_\_  
(sql execute elapsed time)
- c) What is the buffer cache size in the report in the Cache Sizes section of Report Summary? \_\_\_\_\_ (8M)
- d) What does the Load Profile section show for physical reads per second and transaction? \_\_\_\_\_ per second \_\_\_\_\_ per transaction. These numbers will have meaning when you compare them to the next test.
- e) What is the buffer hit % (buffer cache hit ratio) in the Instance Efficiency section? \_\_\_\_\_ (For OLTP type activity, less than 90% is considered a low buffer cache hit ratio.)

### Practice 16-1: Using the DB Cache Advisor (continued)

- f) To confirm the symptoms, check the top misses in the Latch Sleep Breakdown section. Look for misses and sleeps for latches related to the buffer cache. The number of misses and sleeps may be small enough in comparison to the get requests that they would not be considered at all, except that because they are the top in the latch sleep breakdown, they are additional evidence pointing to the buffer cache. What latches are listed at or near the top related to the buffer cache?

---

(You may see “cache buffers lru chain” or “cache buffers chains.”)

8. Use the Buffer Pool Advisory in the AWR report. The navigation path is Back to Top > Advisory Statistics > Buffer Pool Advisory.

What is an appropriate size for the database buffer cache?

\_\_\_\_\_ (16 MB or more)

In the advisory, what is the value shown for the estimated number of physical reads for the recommendation of 16 MB? \_\_\_\_\_ (The change in Est Physical Read Factor should be in the range of 0.5 to 0.03 depending on your machine.)

**Note:** Due to the limitations of the advisor, the maximum estimated cache is only 200% of the current size. A larger buffer pool may provide greater benefit.

9. Change the buffer cache size. To reduce the number of trials, increase DB\_CACHE\_SIZE to 20 MB.
- Navigate to the Memory Advisors page.
    - Click the Database tab.
    - Click the Server tab.
    - Click Memory Advisors in the Database Configuration section.
  - Reduce the shared pool size by 20 MB to 220 MB.
  - Click Apply. This reduced the memory allocated by 20 MB. Now you can increase the buffer cache size.
  - Change the buffer cache to 20 MB, and then click Apply as shown in the following example.

From the previous advisor run, you can see that the optimal size is greater than 16 MB. By changing DB\_CACHE\_SIZE to 20 MB, the next trial should allow the advisor to show a better value.

## Practice 16-1: Using the DB Cache Advisor (continued)

**Memory Advisors**

Page Refreshed **March 30, 2010 10:42:13 AM UTC** [Refresh](#) [Show SQL](#) [Revert](#) [Apply](#)

When Automatic Memory Management is enabled, the database will automatically set the optimal distribution of memory. The distribution of memory will change from time to time to accommodate changes in the workload.

Automatic Memory Management **Disabled** [Enable](#)

**SGA** **PGA**

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.

Automatic Shared Memory Management **Disabled** [Enable](#)

Shared Pool  MB [Advice](#)

Buffer Cache  MB [Advice](#)

Large Pool  MB

Java Pool  MB

Other (MB)

Total SGA (MB) **263** [Calculate](#)

Component	Percentage
Shared Pool	83.5%
Buffer Cache	7.6%
Large Pool	1.5%
Java Pool	4.6%
Other	2.8%

10. Start the workload generator again: `./workgen BC`.

```
$ ./workgen BC
```

11. After about five minutes, stop the workload with the `rm runload` command.

```
$ rm runload
```

12. Create an ADDM report. View the Performance Analysis. (This is the same procedure as shown in step 5.) Check the performance findings. Did the primary issues change?

Yes, the issues have changed. The major change shown on the example report is that Average Active Sessions has reduced from the previous report. Each session in this workload is taking less time.

**Note:** The report you get could be completely different from the example report depending on the classroom machine. Your report may seem to show an increase in issues and impact. A close inspection of the AWR report shows that performance has improved. The example report shows a reduction in the impact of the Top SQL Statements finding and the elimination of the Undersized Buffer Cache finding.

ADDM Performance Analysis

Task Name

ADDM:1239450590\_1\_557 (End Time:Mar 10, 2010 8:31:17 AM)

Filters

View Snapshots

View Report

Task Owner

SYS

Average Active Sessions

0.6

Period Start Time

Mar 10, 2010 8:25:30 AM UTC

Period Duration (minutes)

5.8

Impact (%)

Finding

Occurrences (24 hrs ending with analysis period)

29.4

Top SQL Statements

18 of 42

2.3

"User I/O" wait Class

11 of 42

### Practice 16-1: Using the DB Cache Advisor (continued)

13. On the ADDM page, click View Snapshots. On the Snapshots page, click the Report tab to create an AWR report. Save this report as awr\_BC\_2.html.
14. In another window or browser tab, display the awr\_BC\_1.html file, so that you can compare the two reports.
15. Review the AWR report and identify the primary performance changes.
  - a) In Top 5 Timed Events, the order of events and the amount of time waiting changed. For “db\_sequential\_reads,” the % DB time spent waiting dropped now that more blocks are being held in the cache. Waits in terms of % DB time for the other buffer cache–related items have dropped. Overall, more time is going to useful work (SQL execution) and less time to physical reads. In the following example, almost all the wait time is part of the DB CPU.
 

**Note:** When the elapsed time of the test period is not the same, compare only the metrics that are normalized. Per second, per transaction, and % DB time metrics are normalized metrics.

Top 5 Timed Foreground Events					
Event	Waits	Time(s)	Avg wait (ms)	% DB time	Wait Class
DB CPU		143		71.31	
db file sequential read	89,156	2	0	1.23	User I/O
direct path read	82,810	1	0	0.63	User I/O
direct path write temp	49	1	16	0.39	User I/O
latch: cache buffers chains	19	1	40	0.38	Concurrency

- b) What is the number of physical reads per second shown in the Load Profile section? \_\_\_\_\_

Compare this number with the number of physical reads per second in the AWR\_BC\_1 report. (The number of physical reads per second should have dropped significantly because the buffer cache size is increased to 20 MB.)

- c) What is the buffer hit ratio in the Instance Efficiencies section?  
 \_\_\_\_\_ (OLTP applications expect a buffer hit ratio of 90% or higher.)  
 You should see an improvement from the earlier report.
- d) The Buffer Cache Advisory indicates an optimal buffer cache size. What size is indicated for your instance and workload?  
 \_\_\_\_\_ (A value of approximately 24-36 MB is expected.)

An optimal size is the point where additional memory reduces the Estimated Physical Reads by a small amount. The optimal value for your instance may vary from the expected value.

## Practice 16-2: Using the Keep Pool

In this scenario, you assume the same workload as in scenario 16-1, but you are constrained on memory resources. Adjust the buffer pool memory areas to use as little memory as possible with the most benefit. The `prepare KC` script sets `DB_CACHE_SIZE` to 16 MB and `DB_KEEP_CACHE_SIZE` to 4 MB. A script that assigns a set of frequently used small tables to the keep pool is provided. Will this perform as well or better than having only `DB_CACHE_SIZE` set to 20 MB, thus allowing you to reduce the optimal size of the database buffer cache?

1. Prepare the database for this scenario.
  - a) Log out of Enterprise Manager.
  - b) Run the `./prepare KC` script.

```
$ cd $HOME/workshops
$ ./prepare KC
Database closed.
Database dismounted.
ORACLE instance shut down.
ORACLE instance started.

Total System Global Area  272019456 bytes
Fixed Size                  1335924 bytes
Variable Size              247467404 bytes
Database Buffers           20971520 bytes
Redo Buffers                2244608 bytes
Database mounted.
Database opened.
Finished setup KC
$
```

2. Examine the first AWR report from Practice 16-1, `awr_BC_1.html`. Are there tables that could be cached in a keep pool?
  - a) The Top 5 Timed Foreground Events show one or more of the events that point to the buffer cache size: db file sequential read, db file scattered read, free buffer waits, latch: cache buffer chains, or latch: cache buffer lru chain.

### Top 5 Timed Foreground Events

Event	Waits	Time(s)	Avg wait (ms)	% DB time	Wait Class
DB CPU		257		64.05	
db file sequential read	3,722,448	67	0	16.69	User I/O
latch: cache buffers chains	59	3	51	0.76	Concurrency
direct path read	105,223	2	0	0.60	User I/O
direct path write temp	63	1	19	0.30	User I/O



## Practice 16-2: Using the Keep Pool (continued)

- b) How many full-table scans are being performed? Check the instance activity statistics. **Note:** Table scans (direct read) do not go into the buffer cache; these reads go directly into the session PGA. You should find that the number of table scans (long) and table scans (direct read) are almost the same.

Statistic	Per second	Per transaction
table scans (direct read)		
table scans (long)		
table scans (short)		

- c) Find the SQL statements that have the highest number of physical reads.  
**Hint:** Back to Top > SQL Statistics > SQL Ordered by Reads. The SQL ID may be different on your machine than that shown in the example.

SQL\_ID \_\_\_\_\_

SQL\_ID \_\_\_\_\_

### SQL Ordered by Reads

Physical Reads	Executions	Reads per Exec	%Total	Elapsed Time (s)	%CPU	%IO	SQL Id	SQL Module	SQL Text
2,871,409	6	478,568.17	94.24	139.07	68.44	32.28	9zb56qt165f3z	SQL*Plus	DECLARE CURSOR C2 IS SELECT CU...
2,838,619	60,000	47.31	93.17	121.94	68.13	36.58	431mwkvt65jbb	SQL*Plus	SELECT SUM(AMOUNT_SOLD) TOTAL ...
142,623	96	1,485.66	4.68	16.15	55.09	6.52	cpsda56ma8b15	SQL*Plus	DECLARE CURSOR C1 IS SELECT * ...
135,338	96	1,409.77	4.44	11.24	57.89	8.49	9a9ab05m4mptw	SQL*Plus	SELECT * FROM ALL_TABLES WHERE...
18,344	432	42.46	0.60	5.47	72.07	2.20	6g215w0qfstc2	SQL*Plus	SELECT SUM(UNIT_COST) TOTAL FR...
13,580	432	31.44	0.45	4.47	80.16	3.07	0u9dmyxy318w0	SQL*Plus	SELECT SUM(AMOUNT_SOLD) TOTAL ...
9,011	29	310.72	0.30	89.32	62.47	0.15	14zitd9unpsq9	SQL*Plus	DECLARE CURSOR C2 IS SELECT * ...
5,768	672	8.58	0.19	1.29	60.87	5.40	70uy0ku756w7a	SQL*Plus	SELECT COUNT(INDEX_NAME) FROM ...
5,027	9,251	0.54	0.16	1.33	72.97	6.42	av6kf9w9m362a	SQL*Plus	SELECT COUNT(*) FROM ORDER_ITE...
3,911	5	782.20	0.13	4.80	25.41	3.49	6qvch1xu9ca3g		DECLARE job BINARY_INTEGER := ...

**Practice 16-2: Using the Keep Pool (continued)**

- d) Check some of the SQL IDs for the full text of the statements with the highest number of physical reads to determine the segments involved. Then, find the table names that are being accessed.

What tables are most accessed? \_\_\_\_\_ (SALES, PRODUCTS, and COSTS)

The SQL IDs shown in this example correspond to the SQL IDs in the previous step. Your SQL text will be the same but the SQL ID may differ.

SQL ID	SQL Text
9zb56qt165f3z	<pre> DECLARE CURSOR C2 IS SELECT CUST_ID FROM CUSTOMERS WHERE CUST_ID BETWEEN 0 and 10000; CURSOR C1 IS SELECT PROD_ID FROM PRODUCTS; TYPE emp_list IS TABLE OF VARCHAR2(30); parameters emp_list; j NUMBER(10); tot_amount NUMBER(20); BEGIN FOR C1_ROW in C1 LOOP SELECT SUM(unit_cost) total INTO TOT_AMOUNT FROM sh.costs WHERE prod_id = C1_ROW.prod_id; END LOOP; FOR C2_ROW in C2 LOOP SELECT SUM(AMOUNT_SOLD) total INTO TOT_AMOUNT FROM sh.sales WHERE cust_id = C2_ROW.cust_id; END LOOP; FOR C1_ROW in C1 LOOP SELECT SUM(AMOUNT_SOLD) total INTO TOT_AMOUNT FROM sh.sales WHERE prod_id = C1_ROW.prod_id; END LOOP; END; </pre>
431mwkyt65jbb	<pre> SELECT SUM(AMOUNT_SOLD) TOTAL FROM SH.SALES WHERE CUST_ID = :B1 </pre>

- e) Check the Segments by Physical Reads in the Segment Statistics section.  
What segments have the highest physical reads?

Partitions of the SH.SALES table are expected to appear. This is a very large table. The small tables that should be placed in the keep pool do not appear. Knowledge of the application is needed to properly choose the tables and indexes to keep.

## Practice 16-2: Using the Keep Pool (continued)

- The developers have provided a script named `keep_pool.sh` that will create a small keep pool of 4 MB. Assign some small but very active tables and indexes to the keep pool, and load these tables into the pool. The `keep_pool.sh` script also assigns the most active partitions of the `SALES` table to the keep pool. Review this script, and then run it. The full-table scan caused by the `SELECT * FROM <table_name>` statement will cause the table data to be loaded into the keep pool before the test run. The `ORDER BY` clause is used to force the use of an index so that the index is loaded into the keep cache. These accesses mean that the physical reads due to the first access is not included in the test statistics.

```
$ cat keep_pool.sh
#!/bin/bash
#-- create a KEEP pool
#-- and set the table properties to make use of it
#-- Select statements populate the keep cache

sqlplus -S /nolog <<-EOF
connect / as sysdba
ALTER SYSTEM SET db_keep_cache_size = 4M;
exit
EOF

sqlplus -S /nolog <<-EOF
connect hr/hr

ALTER TABLE EMPLOYEES STORAGE(BUFFER_POOL KEEP);
ALTER TABLE COUNTRIES STORAGE(BUFFER_POOL KEEP);
ALTER TABLE DEPARTMENTS STORAGE(BUFFER_POOL KEEP);
ALTER TABLE LOCATIONS STORAGE(BUFFER_POOL KEEP);
ALTER TABLE REGIONS STORAGE(BUFFER_POOL KEEP);
ALTER TABLE JOB_HISTORY STORAGE(BUFFER_POOL KEEP);
ALTER INDEX HR.DEPT_ID_PK STORAGE(BUFFER_POOL KEEP);
ALTER INDEX HR.DEPT_LOCATION_IX STORAGE(BUFFER_POOL KEEP);
ALTER INDEX HR.JHIST_EMPLOYEE_IX STORAGE(BUFFER_POOL KEEP);

select * from employees;
select * from departments;
select * from countries;
select * from locations;
select * from regions;
select * from job_history;
exit
EOF

sqlplus -S /nolog <<-EOF
connect oe/oe

ALTER TABLE orders STORAGE(BUFFER_POOL KEEP);
ALTER TABLE order_items STORAGE(BUFFER_POOL KEEP);
ALTER TABLE inventories STORAGE(BUFFER_POOL KEEP);
ALTER TABLE customers STORAGE(BUFFER_POOL KEEP);
```

**Practice 16-2: Using the Keep Pool (continued)**

```

ALTER TABLE product_information STORAGE(BUFFER_POOL KEEP);
ALTER INDEX INV_PRODUCT_IX STORAGE(BUFFER_POOL KEEP);
ALTER INDEX PRODUCT_INFORMATION_PK STORAGE(BUFFER_POOL KEEP);
ALTER INDEX OE.ORDER_PK STORAGE(BUFFER_POOL KEEP);
ALTER INDEX OE.ORD_CUSTOMER_IX STORAGE(BUFFER_POOL KEEP);
ALTER INDEX OE.ITEM_ORDER_IX STORAGE(BUFFER_POOL KEEP);
ALTER INDEX OE.PRODUCT_INFORMATION_PK STORAGE(BUFFER_POOL
KEEP);

select * from orders;
select * from order_items;
select * from inventories;
select * from customers
order by customer_id;
select * from product_information
order by product_id;
exit
EOF

sqlplus -S /nolog > /dev/null <<-EOF
set termout off
set feedback off
connect sh/sh
ALTER TABLE SALES MODIFY PARTITION SALES_Q3_1999
STORAGE(BUFFER_POOL KEEP);
ALTER TABLE SALES MODIFY PARTITION SALES_Q4_2001
STORAGE(BUFFER_POOL KEEP);
select * from sales;
ALTER TABLE COSTS STORAGE(BUFFER_POOL KEEP);
select * from costs;
EOF

$ ./keep_pool.sh
$

```

4. Test this new configuration with the workload from scenario 15-1. Run the `./workgen BC` script.

```

$ ./workgen BC

PL/SQL procedure successfully completed.

PL/SQL procedure successfully completed.

$

```

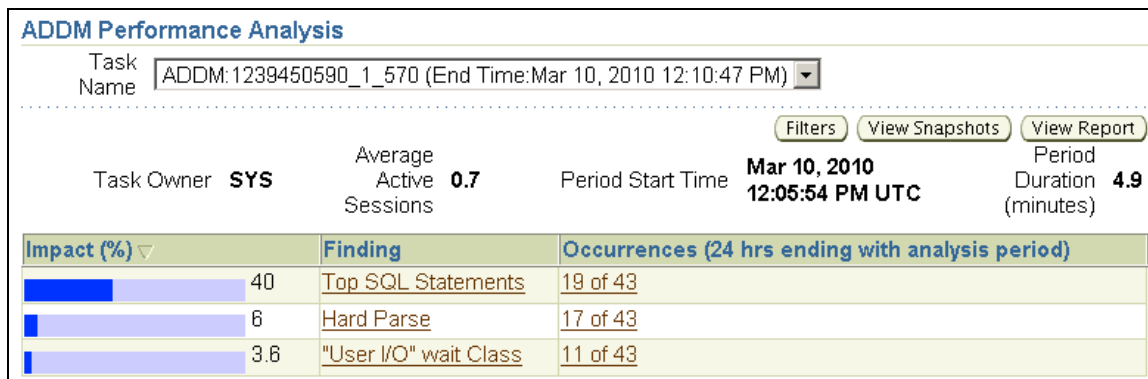
5. Wait for about five minutes, and then end the workload by deleting the runload file. While you are waiting, examine the graphs on the Performance page. Notice the peak values and the I/O per second values.

## Practice 16-2: Using the Keep Pool (continued)

6. Log in to Enterprise Manager. Create an ADDM report. Does the finding “Undersized Buffer Cache” appear? \_\_\_\_\_  
Has the impact for any of the findings changed? \_\_\_\_\_

(Yes. There may be a variety of findings shown, but Undersized Buffer Cache should not be one of them. The Keep Buffer cache may not improve performance and the Undersized Buffer Cache find may remain. The findings and impact numbers may not clearly indicate that performance has improved over the previous ADDM report associated with the second test in Practice 16-1, awr\_BC\_2. In the example, the clearest indicator on the ADDM report is Average Active Sessions. If the workload in two different periods is identical, you can deduce that the time required for each session is reduced, leading to a lower average. If the workload is not identical, this deduction may not hold true. In the following example, you can see that Average Active Sessions has increased somewhat over the second run in practice 16-1.

Recall that impact is measured in % DB Time. There are cases where the Total DB Time goes down, and the DB Time for a particular area does not go down proportionally, so the % DB Time for that area goes up.



7. Create an AWR report. Click View Snapshots. When the Snapshots page appears, click the Reports tab. Save this report as awr\_kc\_1.html.
8. Examine the AWR report and compare it with the previous report, awr\_BC\_2.html.
- Note:** If the elapsed time for awr\_BC\_2.html is very different from the elapsed time for awr\_kc\_1.html, the values of % DB Time and Physical Reads may not show the expected values because of the short durations of the workloads.
- Answer the following questions:
- a) What is the difference in Top 5 Timed Events? (Make the comparison based on % DB Time.) \_\_\_\_\_

The changes in % DB Time may be very small, but it may be enough to change the order of the top waits.

**Practice 16-2: Using the Keep Pool (continued)**

Top 5 Timed Foreground Events					
Event	Waits	Time(s)	Avg wait (ms)	% DB time	Wait Class
DB CPU		123		62.85	
db file sequential read	361,613	6	0	2.95	User I/O
direct path read	21,126	1	0	0.59	User I/O
enq: KO - fast object checkpoint	2	1	393	0.40	Application
latch: cache buffers chains	7	0	39	0.14	Concurrency

- b) In the Load Profile section, is there a difference in physical reads per second and physical reads per transactions for each of the two periods?

	Statistic	Per second	Per transaction
<b>awr_BC_2</b>	<b>Physical reads</b>		
<b>awr_KC_1</b>	<b>Physical reads</b>		

The number of physical reads in the KC\_1 report may be higher than the physical reads reported in BC\_2. This illustrates the efficiency of the buffer cache algorithms. If you recall, the physical reads for the first test were very high; this also illustrates that the keep pool can be an effective way of reducing physical reads when the application access patterns are known.

- c) In the Load Profile, compare DB Time per second for each of the two reports.

	Statistic	Per second
<b>awr_BC_2</b>	<b>DB Time</b>	
<b>awr_KC_1</b>	<b>DB Time</b>	

The DB Time difference is the clearest indicator, in this practice, of a difference in performance.

- d) Is there a difference in Buffer Hit %? \_\_\_\_\_

The overall Buffer Hit % may change very little. The buffer cache hit ratio is a combination of the pools.

- e) Check the Buffer Pool Statistics. Is the keep pool being used?

\_\_\_\_\_

## Practice 16-2: Using the Keep Pool (continued)

Yes, the Buffer Pool Statistics section shows 100% hit ratio on the keep pool. The frequently used blocks are moved to the keep pool and a large number of hits are removed from the default pool calculation. You would expect the hit ratio of the default pool to decline. The statistics over such a small data set and short duration may not show any difference. A small number of physical reads may appear in the Keep Pool statistics; this indicates that the Pool has more blocks assigned to it than there is space available.

- f) How many full-table scans are being performed? Check the instance activity statistics for both reports: awr\_BC\_2 and awr\_KC\_1. (Table scans are unrelated to the physical reads when tables are in the keep buffer cache, but placing indexes in the keep pool can influence the optimizer to use an index rather than do a full-table scan) **Note:** All or nearly all of the full-table scans are direct read. This means that these reads are bypassing the buffer cache and going to PGA (server process memory).

Statistic	Per second (awr_BC_2)	Per second (awr_KC_1)
table scans (long)		
table scans (short)		

9. Considering that this instance is memory constrained, what is the smallest DB\_CACHE\_SIZE that would significantly affect performance based on the Buffer Pool Advisor? \_\_\_\_\_

The advisor shows that there is a benefit by increasing the buffer cache to between 28 and 36 MB. The keep pool in this example did not show improvement in the amount of memory used. This practice did show the process to identify and cache tables in the keep cache.

10. Run the clean up script: From the /home/oracle/workshops directory, execute `./cleanup KC`.

```
$ ./cleanup KC
```

## Practices for Lesson 17

In this practice, you tune PGA Memory.



### **Practice 17-1: Tuning PGA\_AGGREGATE\_TARGET**

You have a decision support system (DSS) that has a variety of queries being performed. Users are complaining that the queries are slow. PGA\_AGGREGATE\_TARGET is set at 20 MB. Determine the smallest setting that eliminates all multipass work areas.

1. Prepare the database for this scenario.
  - a) Log out of Enterprise Manager.
  - b) Execute the prepare script for this practice. Change directory to \$HOME/workshops. Execute ./prepare PGA.

```
$ cd $HOME/workshops
$ ./prepare PGA
Database closed.
Database dismounted.
ORACLE instance shut down.

ORACLE instance started.

Total System Global Area 192507904 bytes
Fixed Size                 1335388 bytes
Variable Size             155193252 bytes
Database Buffers          33554432 bytes
Redo Buffers              2424832 bytes
Database mounted.
Database opened.

Tablespace created.

Database altered.

System altered.

Finished setup PGA
$
```

2. Start the workload generator with ./workgen PGA. This workload consists of several sessions performing a variety of queries that require different sizes of work areas.
3. Log in to Enterprise Manager. Wait a minute or two, and then, while the workload is running, use PGA Advisor and PGA Memory Usage Details to select a new PGA\_AGGREGATE\_TARGET.

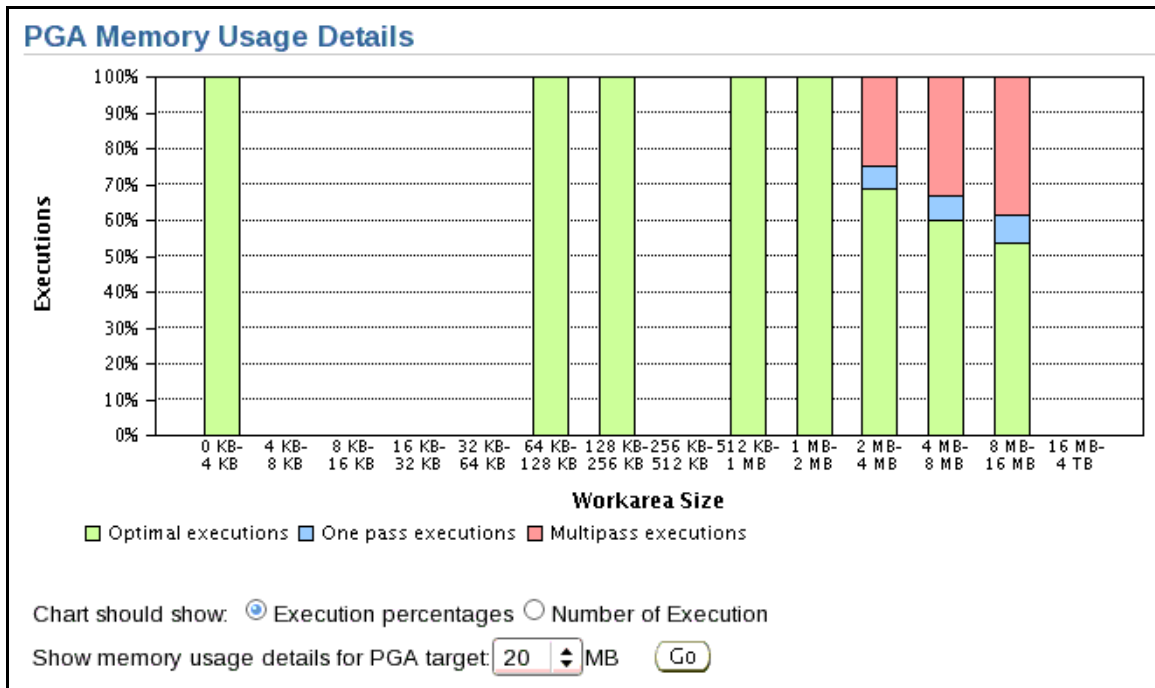
Step	Page	Action
a	Current page	Click the Database tab.

**Practice 17-1: Tuning PGA\_AGGREGATE\_TARGET (continued)**

b	Database home	Click the Server tab.
c	Server page	Click Memory Advisors in the Database Configuration section.
d	Memory Advisors	Click the PGA tab.
e	Memory Advisor (PGA)	Click PGA Memory Usage Details.
f	PGA Memory Usage Details  (See the example screenshot, which follows the table.)	Set the value: Show memory usage details for the PGA target, 20 MB.
		Notice the sizes of the work areas that overflow to disk.
		Notice the relative number of Optimal, One Pass, and Multipass executions.
		Click OK.
g	Memory Advisor (PGA)	Click Advice.
h	PGA Aggregate Target Advice	Notice the overflow range (in red). This is the range where the required PGA for the number of sessions exceeds the PGA Target.
		Make a note of the PGA Target (MB) value where the red line changes to blue. (80 MB is expected.)
		Click Cancel.
i	Memory Advisor (PGA)	Change Aggregate PGA Target to the value of the PGA Target that you noted in step 3h. (80 MB)
		Click Apply.

## Practice 17-1: Tuning PGA\_AGGREGATE\_TARGET (continued)

f) The following screenshot shows the PGA Memory Usage Details page:



4. Stop the workload with `rm runload`.

```
$ rm runload
```

5. Execute the workload with the `./workgen PGA` script.

```
$ ./workgen PGA
```

6. After the workload has been running for about five minutes, stop the workload with `rm runload`.

```
$ rm runload
```

7. Create an AWR snapshot and an AWR report for the last two snapshots.

Step	Page	Action
a	Current page	Click the Database tab.
b	Database home	Click the Server tab.
c	Server tab page	Click Automatic Workload Repository.
d	Automatic Workload Repository	Click the Snapshot link.
e	Snapshots	Click Create.

**Practice 17-1: Tuning PGA\_AGGREGATE\_TARGET (continued)**

f	Confirmation	Click Yes.
g	Snapshots	Select the beginning snapshot number.
		From the Actions drop-down list, select View Report.
		Click Go.
h	View Report: Select Ending Snapshot	Select the ending snapshot number.
		Click OK.

8. Review the AWR report for symptoms and recommendations concerning PGA area sizing. Save the report as awr\_PGA\_1.html.
  - a) Check Top 5 Timed Foreground Events. Is there something that is taking a large amount of % DB Time? \_\_\_\_\_ (direct path read temp, direct path write temp)
  - b) Check Time Model Statistics. What areas are taking the greatest portion of DB Time? \_\_\_\_\_ (sql execute elapsed time, DB CPU). At this point, all that you know is that the SQL is slow, and the main diagnostics agree.
  - c) Determine the SQL with the greatest elapsed time (Back to Top > SQL Statistics > SQL ordered by Elapsed Time). There are several statements that are taking a large amount of elapsed time and CPU time, but very few executions. This combination of elapsed time and CPU time could be caused by excessive reads or sorting. Only one of the top six SQL statements is also in the SQL ordered by Reads, but five out of the top six are also in the SQL ordered by Gets. This indicates that many blocks are being accessed, but most of them are already in the buffer cache. Confirm this by checking the buffer cache hit ratio.

## Practice 17-1: Tuning PGA\_AGGREGATE\_TARGET (continued)

### SQL ordered by Elapsed Time

Elapsed Time (s)	Executions	Elapsed Time per Exec (s)	%Total	%CPU	%IO	SQL Id	SQL Module	SQL Text
557.00	2	278.50	12.51	3.75	0.00	cj4bx12rruv4p	SQL*Plus	select s.prod_id, SUM(s.amount...
423.26	2	211.63	9.51	3.94	0.00	589mxw1dz9mhn	SQL*Plus	select s.prod_id, sum(s.amount...
408.82	7	58.40	9.18	2.51	0.00	8vjh4nbv1phu3	SQL*Plus	select sum(s.AMOUNT_sold) - su...
398.77	12	33.23	8.96	3.43	0.00	2rmw64nrta33j	SQL*Plus	select s.prod_id, sum(s.amount...
387.72	7	55.39	8.71	2.95	0.00	6rz119f0r1wqv	SQL*Plus	select s.prod_id, sum(s.amount...
334.84	18	18.60	7.52	4.27	16.24	54dxhddg0mfdt	SQL*Plus	select * from (select * from d...
309.48	14	22.11	6.95	4.00	18.49	fkmrxxv6fxqar	SQL*Plus	select * from (select * from d...
295.78	12	24.65	6.64	4.28	42.66	bxwvcp4pjr43n	SQL*Plus	select * from (select * from d...
283.28	17	16.66	6.36	5.10	32.50	cssuux75dw74c	SQL*Plus	select * from (select * from d...
275.84	14	19.70	6.20	5.85	44.03	3bg3pfux13pkr	SQL*Plus	select * from (select * from d...
88.01	7	12.57	1.98	2.91	0.36	3g36d1uum3662	SQL*Plus	select PROD_ID, SUM(amount_sol...
63.35	1	63.35	1.42	2.64	0.04	ab7ktcdksu27j	emagent_SQL_oracle_database	/* OracleOEM */ SELECT /*+ IND...
60.30	7	8.61	1.35	1.98	0.01	brtsdaxm7apq7	SQL*Plus	select SUM(amount_sold), PROD_...

d) Navigate to the Instance Efficiencies Percentage report in the Report Summary section (Back to Top > Report Summary). What is the buffer cache hit ratio? \_\_\_\_\_ (99% expected)

e) Check Load Profile for logical reads and physical reads to confirm the buffer cache hit ratio.

Physical reads per second: \_\_\_\_\_

Logical reads per second: \_\_\_\_\_

With a buffer cache hit ratio of near 99%, the physical reads should be about 1% of the logical reads. Is this true? \_\_\_\_\_ (No, the physical reads are much higher.)

There are physical reads that are not counted in the buffer cache hit ratio because the blocks do not go through the buffer cache. These are direct reads.

f) Confirm that direct reads are a large part of the physical reads by checking the Instance Activity Statistics section for physical reads (Main Menu > Instance Activity Statistics). Of the physical reads in the following example, 99% are direct reads. Approximately a half of the direct reads, shown as “physical reads direct,” are to temporary tablespaces, shown as “physical reads direct temporary tablespace.” Physical reads on a temporary tablespace usually indicate a sorting operation and will have nearly matching physical writes. It appears that there may be a problem with temporary space usage, which usually means sort and join operations.

## ***Practice 17-1: Tuning PGA\_AGGREGATE\_TARGET (continued)***

### **Instance Activity Statistics**

physical read IO requests	108,851	357.20	1,943.77
physical read bytes	1,753,186,304	5,753,131.58	31,306,898.29
physical read total IO requests	110,959	364.12	1,981.41
physical read total bytes	1,787,584,512	5,866,010.29	31,921,152.00
physical read total multi block requests	0	0.00	0.00
physical reads	214,012	702.29	3,821.64
physical reads cache	3,186	10.45	56.89
physical reads cache prefetch	425	1.39	7.59
physical reads direct	210,826	691.83	3,764.75
physical reads direct temporary tablespace	103,804	340.64	1,853.64
physical reads prefetch warmup	0	0.00	0.00

- g) Look at the SQL text of the top 5 SQL ordered by Elapsed Time (Back to Top > SQL Statistics > SQL ordered by Elapsed Time). All these statements have a group by or join, or both. These operations force sorts and additional reads if the PGA is not properly sized.

## Practice 17-1: Tuning PGA\_AGGREGATE\_TARGET (continued)

SQL ID	Text
cj4bx12rruv4p	select s.prod_id, SUM(s.amount_sold) - SUM(c.unit_cost) from costs c, sales s where c.prod_id = s.prod_id and s.prod_id = 117 group by s.prod_id
589mxwldz9mhn	select s.prod_id, sum(s.amount_sold) - sum(c.unit_cost) from costs c, sales s where c.prod_id = s.prod_id and s.prod_id = 114 group by s.prod_id
8vjh4nbvlphu3	select sum(s.AMOUNT_sold) - sum(c.unit_cost) from costs c, sales s where c.prod_id = s.prod_id and s.prod_id = 139
2rmw64nrra33j	select s.prod_id, sum(s.amount_sold) - sum(c.unit_cost) from costs c, sales s where c.prod_id = s.prod_id and s.prod_id = 144 group by s.prod_id
6rz119f0rlwqy	select s.prod_id, sum(s.amount_sold) - sum(c.unit_cost) from costs c, sales s where c.prod_id = s.prod_id and s.prod_id = 142 group by s.prod_id

- h) Determine the extra writes that are caused by the current PGA sizing. Check PGA Aggr Summary in the Advisory Statistics section (Back to Top > Advisory Statistics > PGA Aggr Summary).

What is the PGA Cache Hit ratio? \_\_\_\_\_ (25 in the example; your numbers may vary)

What is the Extra W/A (work area) MB Read/Written? \_\_\_\_\_ (3,600 MB in the example). The example indicates that 3,600 MB were read and written because the work area could not be completed in memory.

### PGA Aggr Summary

PGA Cache Hit %	W/A MB Processed	Extra W/A MB Read/Written
25.46	1,242	3,634

## Practice 17-1: Tuning PGA\_AGGREGATE\_TARGET (continued)

- i) Determine the recommended PGA sizing from the AWR report. In the Advisories section, find the PGA Memory Advisory (Back to Top > Advisory Statistics > PGA Memory Advisory). To find the minimum size with no multipass work areas, choose the smallest value in which Estd PGA Overalloc Count is 0. Using the example, the size is 128 MB. The estimated extra work area read from and written to disk does not drop significantly until the PGA is 128 MB. Your results may vary.

### PGA Memory Advisory

PGA Target Est (MB)	Size Factr	W/A MB Processed	Estd Extra W/A MB Read/ Written to Disk	Estd PGA Cache Hit %	Estd PGA Overalloc Count	Estd Time
10	0.13	1,832.20	3,679.95	33.00	81	29,829,284
20	0.25	1,832.20	3,679.95	33.00	81	29,829,284
40	0.50	1,832.20	3,679.95	33.00	81	29,829,284
60	0.75	1,832.20	3,679.95	33.00	81	29,829,284
80	1.00	1,832.20	2,692.35	40.00	80	24,484,838
96	1.20	1,832.20	2,689.44	41.00	53	24,469,095
112	1.40	1,832.20	2,439.99	43.00	13	23,119,177
128	1.60	1,832.20	1,148.66	61.00	0	16,131,062
144	1.80	1,832.20	744.64	71.00	0	13,944,711
160	2.00	1,832.20	452.94	80.00	0	12,366,113
240	3.00	1,832.20	0.00	100.00	0	9,915,029
320	4.00	1,832.20	0.00	100.00	0	9,915,029
480	6.00	1,832.20	0.00	100.00	0	9,915,029
640	8.00	1,832.20	0.00	100.00	0	9,915,029

- j) Observe the number of multipass and one-pass work areas that are being used. Find PGA Aggr Target Histogram in the Advisories section of the AWR report (Back to Top > Advisory Statistics > PGA Aggr Target Histogram). The work areas that cannot be contained in memory write to disk (temporary tablespace) and are recorded as one-pass or multipass work areas. In PGA Aggr Target Histogram, the column on the right shows the minimum requested size for a one-pass execution. The example shows a number of multipass executions and one-pass executions for sorts larger than 2 MB.

Low Optimal	High Optimal	Total Execs	Optimal Execs	1-Pass Execs	M-Pass Execs
2K	4K	1,082	1,082	0	0
64K	128K	4	4	0	0
256K	512K	3	3	0	0
512K	1024K	330	330	0	0
1M	2M	2	2	0	0
2M	4M	38	0	8	30
4M	8M	62	0	4	58
8M	16M	56	2	0	54

9. Make a new setting for PGA\_AGGREGATE\_TARGET on the basis of the value that you determined from the PGA Memory Advisory. The goal is to eliminate multipass executions. In a terminal window, connect as sysdba and change PGA\_AGGREGATE\_TARGET to 128 MB.

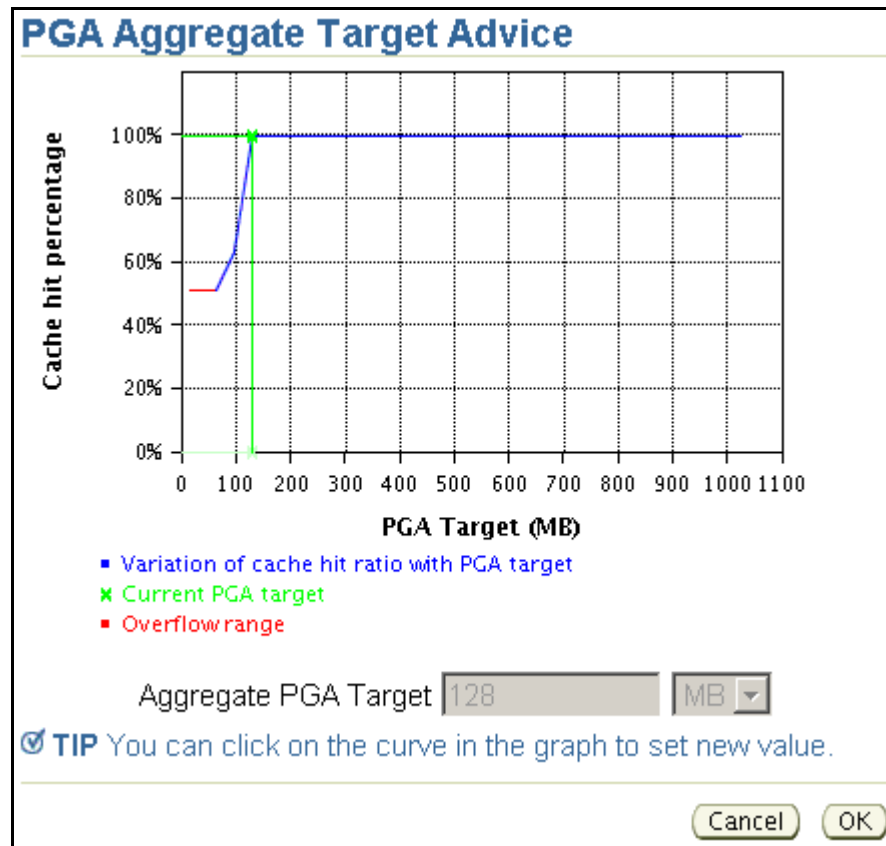


## Practice 17-1: Tuning PGA\_AGGREGATE\_TARGET (continued)

```
$ sqlplus / as sysdba
SQL> ALTER SYSTEM SET PGA_AGGREGATE_TARGET = 128M;
SQL> exit;
```

10. Execute the workload with the `./workgen PGA` script.
11. After a few minutes, view the Memory Advisors page, PGA tab (Server tab > Memory Advisors > PGA tab). Check the PGA Advisor and the PGA Memory Usage Details pages.
  - a) View the PGA Advice page. Have the recommended values changed?

Yes, the advice graph has changed; the following example indicates that a value of approximately 130 MB is optimal.



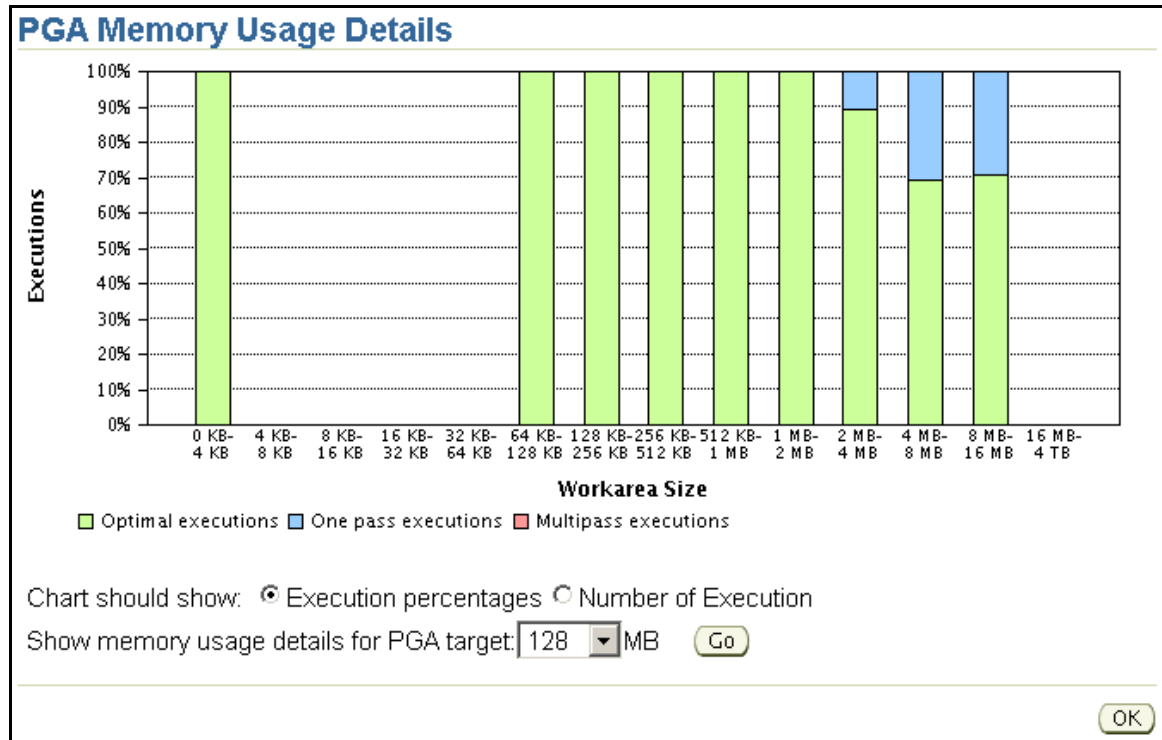
- b) View the PGA Memory Usage Details page.

When the PGA Memory Usage Details page appears, it starts with the lowest possible PGA Target value. Set the value to be displayed to your current value of `PGA_AGGREGATE_TARGET`.

Are there any multipass work areas?

**Answer:** No. The PGA Memory Usage Details histogram does not show any multipass work areas at 128 MB.

## Practice 17-1: Tuning PGA\_AGGREGATE\_TARGET (continued)



12. After the workload has been running for about five minutes, stop the workload with `rm runload`.

```
$ rm runload
```

13. Create an AWR snapshot and an AWR report for the last two snapshots. Use the same procedure as was used in step 6.
14. Review the AWR report for symptoms, recommendations concerning PGA area sizing, and the changes that show improved performance.
  - a) Save this report for later viewing: Click “Save to File.” Name the file `awr_PGA_2.html`.
  - b) Check Load Profile for physical reads and writes per second. The physical reads and writes per second have dropped significantly in the example compared to the previous report `awr_PGA_1.html`.
  - c) Have the SQL statements that have the highest elapsed time changed? Find the SQL statements identified in step 6c that have the greatest elapsed time or compare the two reports side by side. The order may have changed, but the same SQL IDs are in the Top SQL. The SQL ID is the hash value of the SQL statement and does not change between runs. Use the SQL ID to find the statements. Notice that the reads per execution have dropped significantly. Users would notice that the statements are responding more quickly based on this example.

## Practice 17-1: Tuning PGA\_AGGREGATE\_TARGET (continued)

SQL ordered by Elapsed Time

Elapsed Time (s)	Executions	Elapsed Time per Exec (s)	%Total	%CPU	%IO	SQL Id	SQL Module	SQL Text
488.05	2	244.02	13.29	3.52	0.00	<a href="#">cj4bx12rruv4p</a>	SQL*Plus	select s.prod_id, SUM(s.amount...
382.26	13	29.40	10.41	3.84	0.00	<a href="#">2rmw64nrra33j</a>	SQL*Plus	select s.prod_id, sum(s.amount...
352.73	18	19.60	9.61	3.89	2.98	<a href="#">54dxhddg0mfdt</a>	SQL*Plus	select * from (select * from d...
349.68	8	43.71	9.52	3.32	0.00	<a href="#">6vjh4nbv1phu3</a>	SQL*Plus	select sum(s.AMOUNT_sold) - su...
348.84	7	49.85	9.50	3.25	0.00	<a href="#">6rz119f0r1wgy</a>	SQL*Plus	select s.prod_id, sum(s.amount...
327.37	13	25.18	8.92	3.45	30.75	<a href="#">fkmrxw6fxqar</a>	SQL*Plus	select * from (select * from d...
318.64	16	19.92	8.68	4.11	7.57	<a href="#">cssuux75dw74c</a>	SQL*Plus	select * from (select * from d...
301.11	12	25.09	8.20	4.14	21.73	<a href="#">3bq3pfux13pkr</a>	SQL*Plus	select * from (select * from d...
276.71	14	19.77	7.54	4.67	20.49	<a href="#">bxwvcp4pjr43n</a>	SQL*Plus	select * from (select * from d...
102.54	2	51.27	2.79	6.07	0.00	<a href="#">589mxw1dz9mhn</a>	SQL*Plus	select s.prod_id, sum(s.amount...
62.99	7	9.00	1.72	4.08	0.02	<a href="#">3g36d1uum3662</a>	SQL*Plus	select PROD_ID, SUM(amount_sol...
43.50	8	5.44	1.18	3.12	3.65	<a href="#">8rtsdaxm7apq7</a>	SQL*Plus	select SUM(amount_sold), PROD_...

- d) Determine the improvement in the PGA cache hit percentage. How could the extra write statistic be deceiving? PGA Cache Hit % has increased, showing that more sorting was done in memory. Extra Work Area MB Read/Written could show an increase in I/O from the previous report because the statistic is a total over the report interval (especially if the durations of the reports are different).

### PGA Aggr Summary

PGA Cache Hit %	W/A MB Processed	Extra W/A MB Read/Written
63.25	1,141	663

- e) Determine the recommended PGA sizing from the AWR report PGA Memory Advisory. Following the process of finding the first value with the Estd PGA Overalloc Amount equal to 0, the recommended value should still be the same as in Step 8i.

## Practice 17-1: Tuning PGA\_AGGREGATE\_TARGET (continued)

### PGA Memory Advisory

PGA Target Est (MB)	Size Factr	W/A MB Processed	Estd Extra W/A MB Read/ Written to Disk	Estd PGA Cache Hit %	Estd PGA Overalloc Count	Estd Time
16	0.13	1,148.33	2,265.97	34.00	63	20,715,288
32	0.25	1,148.33	2,265.97	34.00	63	20,715,288
64	0.50	1,148.33	2,265.97	34.00	63	20,715,288
96	0.75	1,148.33	2,193.14	34.00	46	20,273,418
128	1.00	1,148.33	331.44	78.00	0	8,978,093
154	1.20	1,148.33	331.44	78.00	0	8,978,093
179	1.40	1,148.33	238.44	83.00	0	8,413,807
205	1.60	1,148.33	79.08	94.00	0	7,446,916
230	1.80	1,148.33	0.00	100.00	0	6,967,150
256	2.00	1,148.33	0.00	100.00	0	6,967,150
384	3.00	1,148.33	0.00	100.00	0	6,967,150
512	4.00	1,148.33	0.00	100.00	0	6,967,150
768	6.00	1,148.33	0.00	100.00	0	6,967,150
1,024	8.00	1,148.33	0.00	100.00	0	6,967,150

f) Were there any multipass work areas in the reported period?

No multipass work areas are reported. The number of one-pass work areas cannot be directly compared due to the differences in time periods of the reports and the multipass sorts that are now one-pass. There is the additional complication that because the system is more efficient, more statements—and thus more sorts—can be performed in the same period.

## Practices for Lesson 18

The goal of this practice is to use the Automatic Memory Tuning capability of Oracle Database 11g.

## Practice 18-1: Enabling Automatic Shared Memory

In this practice, you enable Automatic Shared Memory Tuning. The initial configuration is simulating a memory-constrained system. The buffer cache is 16 MB, the keep pool is 4 MB, and the shared pool is 132 MB. The shared pool is loaded with kept objects as in a previous practice. For this practice, `SGA_MAX_SIZE` is set to 400 MB and `MEMORY_MAX_TARGET` is set to 800 MB so that you can easily determine the SGA optimal size. For this practice, you are using the Automatic Shared Memory Management feature of Oracle Database.

1. Prepare the instance for this practice.
  - a) Log out of Enterprise Manager.
  - b) Run the `./prepare ASMM` script.

```
$ ./prepare ASMM
Database closed.
Database dismounted.
ORACLE instance shut down.

ORACLE instance started.

Total System Global Area  313860096 bytes
Fixed Size                  1336232 bytes
Variable Size              289410136 bytes
Database Buffers           16777216 bytes
Redo Buffers                6336512 bytes
Database mounted.
Database opened.
Finished setup ASMM
$
```

- c) Enable the Automatic Shared Memory Management feature in Enterprise Manager. Assume that SGA memory is constrained to 400 MB.

Step	Page	Action
a		Log in to Enterprise Manager.
b	Database home	Click the Server tab.
c	Server subpage	Click Memory Advisors.
d	Memory Advisors (See the example screenshot, which follows the table.)	Verify that Maximum SGA Size (MB) is set to 400 MB.
		Click Enable for Automatic Shared Memory Management.

## Practice 18-1: Enabling Automatic Shared Memory (continued)

e	Enable Automatic Shared Memory Management	The memory size displayed is the sum of the current pool settings. Change this value to 168 MB. (The value must be evenly divisible by 4.) Note that smaller values of Total SGA Size (MB) will cause the Total SGA Size to default to the Maximum SGA Size.
		Click OK.
f	Memory Advisors	A confirmation message is shown.

d) The following screenshot shows the Memory Advisors page:

**Memory Advisors**

Page Refreshed **May 2, 2010 7:31:57 PM UTC** [Refresh](#) [Show SQL](#) [Revert](#) [Apply](#)

When Automatic Memory Management is enabled, the database will automatically set the optimal distribution of memory. The distribution of memory will change from time to time to accommodate changes in the workload.

Automatic Memory Management **Disabled** [Enable](#)

**SGA** [PGA](#)

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.

Automatic Shared Memory Management **Disabled** [Enable](#)

Shared Pool  MB [Advice](#)

Buffer Cache  MB [Advice](#)

Large Pool  MB

Java Pool  MB

Other (MB)

Total SGA (MB) **159** [Calculate](#)

**Maximum SGA Size**

The Maximum SGA Size specifies the maximum memory that the database may allocate. If you specify the Maximum SGA Size, you can later dynamically change SGA component sizes (provided the total SGA size does not exceed the Maximum SGA Size).

Maximum SGA Size (MB)

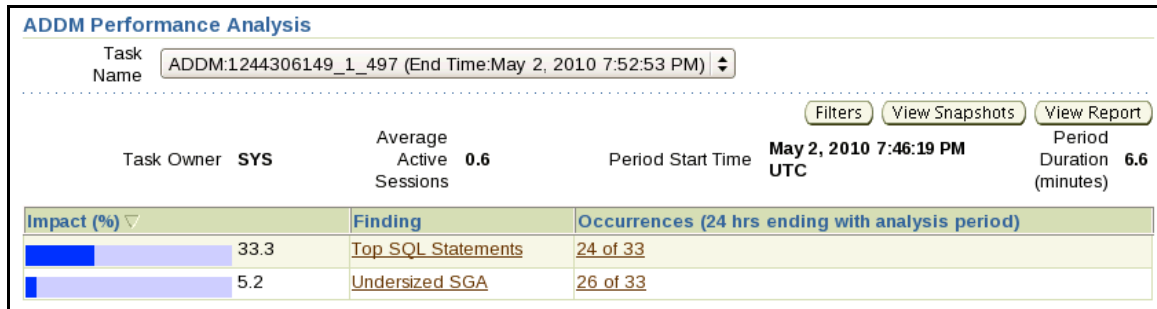
The database must be restarted before any changes to this value take effect.

2. Run the workload generator for Practice 16-1 again: `./workgen BC`.

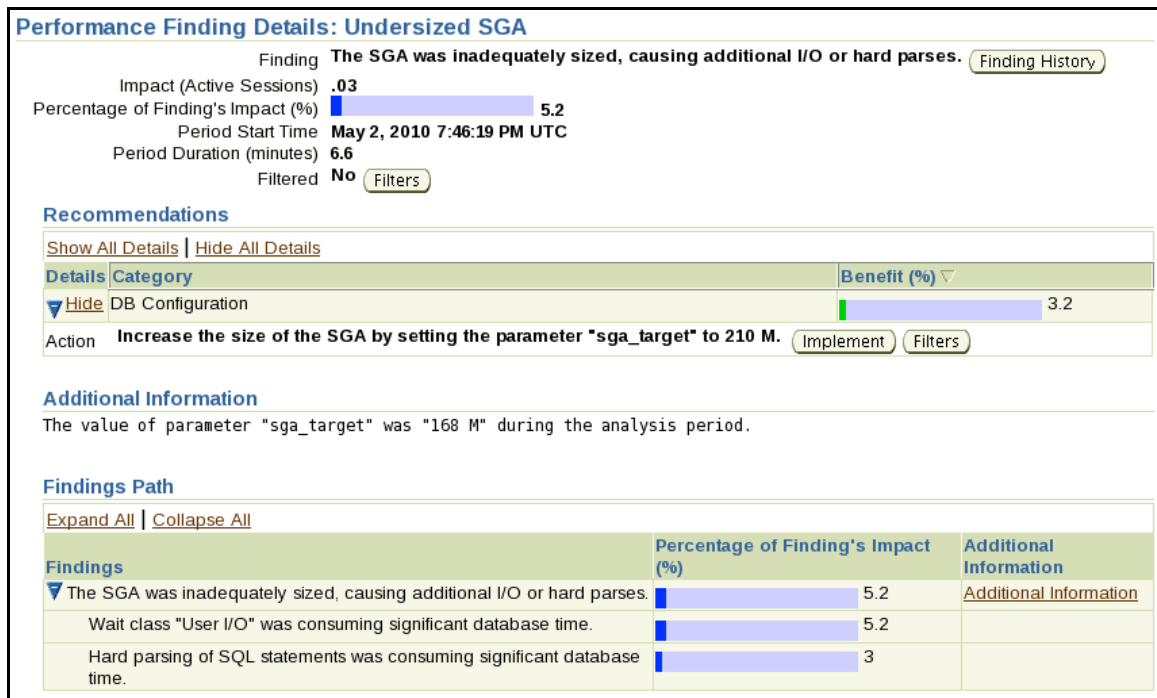
```
$ ./workgen BC
```

- Log in to EM and navigate to the Performance tabbed page. After about five minutes, run an ADDM report. For any findings related to memory sizing, drill down to the details.
  - View the ADDM report. There may be findings that are related to SQL statements and CPU usage, such as "CPU Usage" or "Top SQL by DB Time." There should be an "Undersized SGA" finding.

## Practice 18-1: Enabling Automatic Shared Memory (continued)



- b) View the finding details for “Undersized SGA” and view the recommendation. What is the recommended size for the SGA? \_\_\_\_\_  
(The recommendation will vary.)



- c) Return to the ADDM page.



### Practice 18-1: Enabling Automatic Shared Memory (continued)

d) On the ADDM page, click View Snapshots, and then click the Report tab.

In the Report Summary section, examine the cache sizes.

What are the start and end sizes for the buffer cache and shared pool?

Cache	Begin	End
Buffer Cache		
Shared Pool Size		

The values did not change because the sizes are minimum sizes specified by the initialization parameters for these pools.

4. View the Memory Advisors page (Server tab > Memory Advisors). What are the current memory parameter settings?
  - a) The memory parameter settings may vary depending on the machine that you are using.
 

Buffer Cache \_\_\_\_\_ (16 MB)

Shared Pool \_\_\_\_\_ (120 MB)

Large Pool \_\_\_\_\_ (4 MB)

Java Pool \_\_\_\_\_ (12 MB)

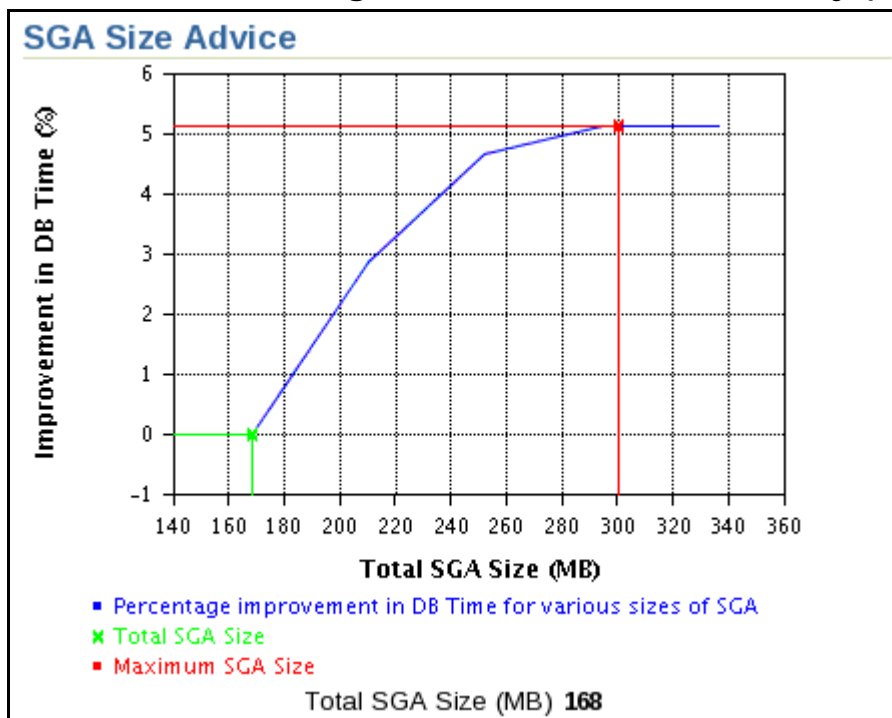
Other \_\_\_\_\_ (16 MB)

**Note:** The EM Memory Advisors page does not show the value of the keep pool separately. It is included in Other.
5. Check Total SGA Size Advisory for an optimal SGA size. Click Total SGA Size (MB) Advice on the Memory Advisors page. The advisory shows that increasing the SGA size gives some improvement in performance.

What is the recommended size? \_\_\_\_\_

In the following example, the greatest benefit for the memory added will be seen at approximately 210 MB, the first inflection point on the graph, but additional benefit will be gained until approximately 250 MB, the second inflection point, when there is little or no more benefit for adding memory.

## Practice 18-1: Enabling Automatic Shared Memory (continued)



6. Stop the workload with the `rm runload` command.

```
$ rm runload
```

7. On the Memory Advisors page, change Total SGA Size to 260 MB. Click Apply.
8. Observe the initial settings for the SGA on the Memory Advisors page. (Click Refresh to view the new distribution of memory.)

What is the value of the buffer cache? \_\_\_\_\_

What is the value of the shared pool size? \_\_\_\_\_

The values for the shared pool and buffer cache may vary based on the machine you are using.

9. Start the workload again: `./workgen BC`.

```
$ ./workgen BC
```

10. On the Performance page, what are the differences between this and the previous run?

The Average Active Sessions graph and the Load average graphs may show small differences. The I/O graph shows a large difference in the number of buffer cache reads.

11. Observe the current settings for the SGA on the Memory Advisors page. (Click Refresh to view any changes in the distribution of memory.)

What is the value of the buffer cache? \_\_\_\_\_

What is the value of the shared pool size? \_\_\_\_\_

## Practice 18-1: Enabling Automatic Shared Memory (continued)

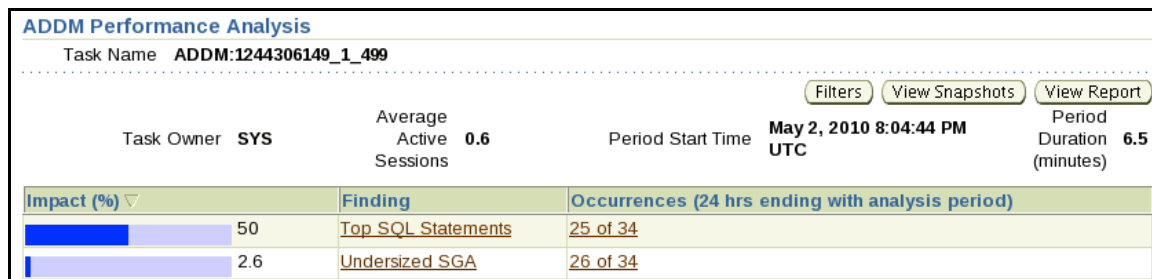
The values for the shared pool and buffer cache may vary based on the machine you are using. It is expected that the values will stay the same as the initial change seen in step 8.

12. After about five minutes, stop the workload with `rm runload`.

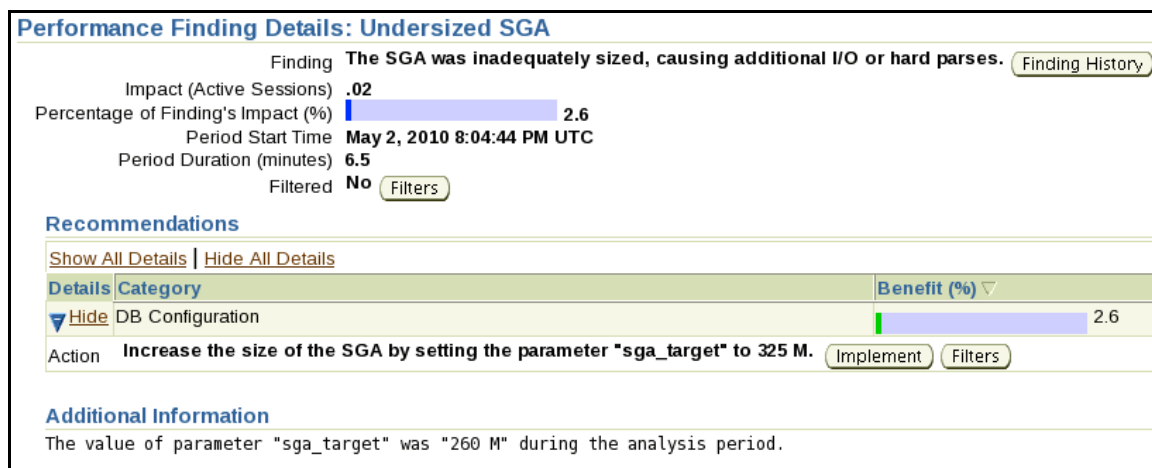
```
$ rm runload
```

13. Click Run ADDM Now on the Performance tabbed page to run an ADDM report. View the results and drill down to the detail pages.

- a) The number of findings and the impact of the issues may be misleading. It is common that correcting memory issues has a much greater impact than is estimated. It is possible that there are more findings and Impact % may be larger. Notice that the number of Average Active Sessions is reduced even though the load was identical. This indicates that the instance is running more efficiently.



- b) The finding “Undersized SGA” may still be reported. In the example, the recommendation is to increase the SGA to 325 MB. The impact of this issue as shown in the screenshot is only 2.6%. The recommended size may range to a much higher value.



- c) A careful comparison of the AWR reports from this and the previous runs will show an improvement. In the load profile, you may see a small improvement in DB Time per second, and a significant reduction in physical reads.

14. What would be a reasonable next tuning step based on the ADDM report?

## ***Practice 18-1: Enabling Automatic Shared Memory (continued)***

**Note:** The number and ranking of the issues shown in the ADDM reports of various machines will vary.

- a) **Hard Parse:** This is likely to be caused by an undersized shared pool. This is confirmed by the Undersized SGA finding in the preceding screenshot. Increase the SGA size if allowed by memory constraints.
- b) **CPU Usage:** The hard-parse activity uses CPU. Implementing a solution for the hard-parse finding will impact CPU usage.
- c) **Soft Parse:** This is primarily an application issue. Check the `SESSION_CURSOR_CACHE` value and set it to a higher value if it is already set.
- d) **The sizing of the SGA** has an impact on a variety of issues, which is usually larger than the impact rating assigned to it. Therefore, it is reasonable to assume that if the SGA is increased again, there will be further improvements in performance.

## Practice 18-2: Adjusting Memory as Workloads Change

On the basis of the recommendation of the ADDM report in the previous scenario, the SGA memory constraint was raised to 260 MB. You have been using an allocation of 200 MB of memory for the PGA work areas. The current application has seen an increase in activity, and now another application is being added to the database. With it comes an additional allocation of instance memory. Make use of Automatic Memory Management to allocate this additional memory for best performance.

1. You are given an absolute maximum of 800 MB of instance memory and are instructed to use as little as required to get reasonable performance. Set the maximum memory size to 800 MB. Enable Automatic Memory Management. Set the Memory target to 500 MB. Note by setting the value larger than needed during testing, you can find the optimal value, and then lower the setting. To set the parameters to enable AMM, you disable ASMM and remove the SGA\_TARGET value.
  - a) On the Memory Advisors page in the Current Allocation section, disable Automatic Shared Memory Management. Click Disable.
  - b) On the Disable Automatic Shared Memory Management page, change the Buffer Cache to 16 MB, so that the value will not impose a minimum.
  - c) Click Calculate; the Total SGA value should change.
  - d) Click OK.
  - e) On the Memory Advisors page, a Confirmation message saying that ASMM has been disabled appears.
2. On the Memory Advisors page, enable Automatic Memory Management. Click Enable.

The screenshot shows the 'Memory Advisors' page. At the top, it says 'Page Refreshed March 12, 2010 9:40:27 AM UTC' with buttons for 'Refresh', 'Show SQL', 'Revert', and 'Apply'. Below this, a message states: 'When Automatic Memory Management is enabled, the database will automatically set the optimal distribution of memory. The distribution of memory will change from time to time to accommodate changes in the workload.' A red box highlights the 'Automatic Memory Management Disabled' status with an 'Enable' button next to it. At the bottom, there are tabs for 'SGA' and 'PGA', with 'PGA' currently selected.

- a) On the Enable Automatic Memory Management page, set the maximum memory size to 800 MB and the Total Memory Size for Automatic Memory Management to 500 MB.

## Practice 18-2: Adjusting Memory as Workloads Change (continued)

**Enable Automatic Memory Management**

When Automatic Memory Management is enabled, the database will automatically set the optimal distribution of memory. The distribution of memory will change from time to time to accommodate changes in the workload. The Maximum Memory Size specifies the maximum memory that the database may allocate and must be set in order to use Automatic Memory Management.

Current Memory Usage (PGA+SGA) (MB) 460

Maximum Memory Size 800 MB

Changing the maximum memory size requires a restart of the database.

Total Memory Size for Automatic Memory Management 500 MB

CancelOK

- b) Navigate to the Memory Advisors page. (The allocation of the buffer cache and the shared pool memory will vary.)

The Total SGA Size is \_\_\_\_\_ MB

The buffer cache is \_\_\_\_\_ MB.

The shared pool is \_\_\_\_\_ MB.

- c) Click the PGA tab. What is the value of the current PGA allocated memory?  
\_\_\_\_\_ (Current PGA Allocated (KB))

3. Start the workload generator. The new workload is simulated by:  
./workgen AMM

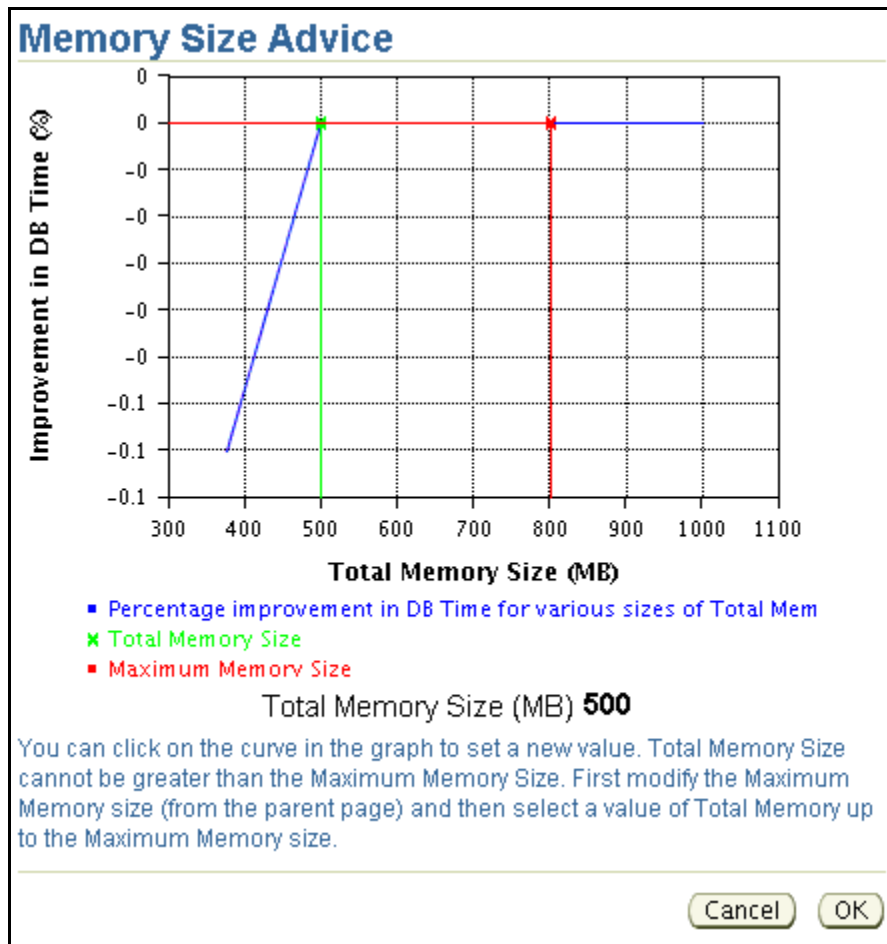
```
$ ./workgen AMM
```

4. Observe the Memory advisor graphs for about three minutes. Be sure to click Refresh.
- a) Did the memory settings change? \_\_\_\_\_
- Note:** These values change during the course of the workload.
- b) What is the value of the buffer cache? \_\_\_\_\_
- c) What is the value of the shared pool? \_\_\_\_\_
- d) What is the value of the PGA? \_\_\_\_\_ (Current PGA Allocated (KB) changes.)
- e) Click Advice. View the Memory Size Advice Graph. What would be a better value of Total Memory? \_\_\_\_\_

In the following example, the graph shows no improvement when memory is increased to more than 500 MB.

**Note:** The graph produced by your machine may vary.

## Practice 18-2: Adjusting Memory as Workloads Change (continued)



- On the Memory Advisors page, change the Total Memory Size to 750 MB and click Apply.
- Observe the Memory advisor graphs for about two or three minutes.
  - Did the memory settings change? \_\_\_\_\_

**Note:** These values can change during the course of the workload.

  - What is the value of the buffer cache? \_\_\_\_\_
  - What is the value of the shared pool? \_\_\_\_\_
  - What is the value of the PGA? \_\_\_\_\_
  - Click Advice. View the Memory Size Advice graph. What would be a better value of Total Memory? \_\_\_\_\_  
At this point, the graph should show that there is little or no advantage to increasing the total memory.
- Stop the workload with `rm runload`.

```
$ rm runload
```

## ***Practice 18-2: Adjusting Memory as Workloads Change (continued)***

8. In this practice, the SGA changed very little. It may not have changed at all on some machines. The PGA, however, was being used by the new workload and the distribution of memory between the PGA and SGA was affected.
9. How much tuning was required to find an optimum memory size for this workload?

The tuning did not require a shutdown after the maximum allowed memory value was set. The Advice graph gave a clear indication of the optimal size, so the number of iterations was reduced.

10. Run the `./cleanup AMM` script.

```
$ ./cleanup AMM
```



## Practices for Lesson 19

There are no practices for this lesson.

## Practices for Lesson 20

There are no practices for this lesson.

## Practices for Appendix B

This practice is optional and may be used with Appendix B.

## Practice for Appendix B-1: Installing Statspack

The goal of this practice is to install the Statspack repository and packages in the `orcl` database. For this exercise, install the Statspack repository in the `SYSAUX` tablespace. The Statspack repository may be installed in any permanent tablespace. Make sure that you use the `perfstat` password for the `PERFSTAT` user. The `sp*.sql` scripts are located in the `$ORACLE_HOME/rdbms/admin` directory.

1. To prepare the database for this practice, connect with `/ as SYSDBA` to drop the Statspack repository. Use the `spdrop.sql` script located in the `$ORACLE_HOME/rdbms/admin` directory.

```
$ sqlplus / as sysdba

SQL> @?/rdbms/admin/spdrop.sql

SQL>
```

2. Still connected as `SYSDBA` using `SQL*Plus`, execute the `spcreate.sql` script to create the Statspack repository and packages.

The script prompts you for the name and password of the owner of the Statspack object. Make the username `perfstat` with a password of `perfstat`. The default tablespace for the Statspack objects is `SYSAUX`. The default tablespace for the `perfstat` user is `TEMP`.

```
SQL> @?/rdbms/admin/spcreate.sql
```

3. Make sure that you did not have errors while installing Statspack. Check the log files that were generated during installation. The following is a command-line solution:

```
SQL> host ls *.lis
spcpkg.lis  spctab.lis  spcusr.lis  spdtab.lis  spdusr.lis

SQL> host grep error *.lis
spcpkg.lis:No errors.
spcpkg.lis:No errors.
spcpkg.lis:SPCPKG complete. Please check spcpkg.lis for any
errors.
spctab.lis:SPCTAB complete. Please check spctab.lis for any
errors.
spcusr.lis:SPCUSR complete. Please check spcusr.lis for any
errors.
spdtab.lis:SPDTAB complete. Please check spdtab.lis for any
errors.
spdusr.lis:SPDUSR complete. Please check spdusr.lis for any
errors.

SQL> exit
```

## Practice for Appendix B-2: Creating Snapshots

The goal of this practice is to run a workload on the `orcl` database and to capture Statspack snapshots for that workload.

1. Before you capture your first Statspack snapshot, execute the `lab_21_02_01.sh` script located in your `$HOME/labs` directory. This script creates a tablespace and a user as the setup for the rest of the practice.

```
$ cd $HOME/labs

$ ./lab_21_02_01.sh
...
SQL> Connected.
SQL> SQL> SQL> SQL> drop tablespace tbsspc including contents
and datafiles
*
ERROR at line 1:
ORA-00959: tablespace 'TBSSPC' does not exist

SQL> SQL>      2      3      4      5
Tablespace created.

SQL> SQL>
PL/SQL procedure successfully completed.

SQL> SQL> drop user spc cascade
*
ERROR at line 1:
ORA-01918: user 'SPC' does not exist

SQL> SQL>      2      3
User created.

SQL> SQL>
Grant succeeded.

SQL> SQL> Connected.
SQL> SQL> drop table spct purge
*
ERROR at line 1:
ORA-00942: table or view does not exist

SQL>
Table created.

SQL> SQL> > >
PL/SQL procedure successfully completed.

SQL> SQL> Disconnected ...
```

## Practice for Appendix B-2: Creating Snapshots (continued)

```
$
```

2. Create a level 7 Statspack snapshot. Make sure that you retrieve the corresponding snapshot ID. Set the `i_snap_level` parameter of the `statspack.snap` procedure to 7. This script is `$HOME/solutions/sol_21_02_02.sh`.

```
$ sqlplus /nolog
SQL> CONNECT perfstat/perfstat

SQL> variable snap number;
SQL> begin
  2  :snap := statspack.snap(i_snap_level=>7);
  3  end;
  4  /

PL/SQL procedure successfully completed.

SQL> print snap

          SNAP
-----
           1

SQL> exit
```

3. Execute the `start_21_02_03.sh` script from the `labs` directory. This script starts the workload on your `orcl` database. The script takes about one minute to execute.

```
$ cd $HOME/labs
$ ./start_21_02_03.sh
PL/SQL procedure successfully completed.

PL/SQL procedure successfully completed.

PL/SQL procedure successfully completed.
...
PL/SQL procedure successfully completed.

PL/SQL procedure successfully completed.

PL/SQL procedure successfully completed.
$
```

## ***Practice for Appendix B-2: Creating Snapshots (continued)***

4. After the workload finishes running, capture a new level 7 Statspack snapshot. Make sure that you retrieve the corresponding snapshot ID. The number is system-assigned and is not controllable. The snap numbers are not necessarily consecutive.

```
$ cd $HOME/labs
$ sqlplus perfstat/perfstat
...

SQL> variable snap number;
SQL> begin
  2     :snap := statspack.snap(i_snap_level=>7);
  3 end;
  4 /

PL/SQL procedure successfully completed.

SQL> print snap
      SNAP
-----
          2

SQL> exit
...
$
```

## Practice for Appendix B-3: Generating Statspack Reports

The goal of this practice is to generate a Statspack report, examine it, and fix any issues that you discover while interpreting the Statspack report.

1. Generate a Statspack report between the two previously captured snapshots. Use the `spreport.sql` script. Be sure to use the snap numbers that you found in the previous practice. When prompted for a file name, use `sp_1_2.lst`.

```
$ cd $HOME/labs
$ sqlplus perfstat/perfstat
...
SQL> set echo on

SQL> @?/rdbms/admin/spreport
...
Listing all Completed Snapshots
```

Instance Comment	DB Name	Snap Id	Snap Started	Snap Level
orcl	ORCL	1	16 Mar 2010 09:54	7
		2	16 Mar 2010 09:57	7

```

Specify the Begin and End Snapshot Ids
~~~~~
Enter value for begin_snap: 1
Begin Snapshot Id specified: 1

Enter value for end_snap: 2
End Snapshot Id specified: 2

Specify the Report Name
~~~~~
The default report file name is sp_1_2. To use this name,
press <return> to continue, otherwise enter an alternative.

Enter value for report_name: sp_1_2.lst
...
sga_max_size          419430400
sga_target            0
shared_pool_size      0
spfile
/u01/app/oracle/product/11.2.0/dbhome_1/dbs/spfileorcl.ora
streams_pool_size     4194304
undo_tablespace        UNDOTBS1
~~~~~

```



## Practice for Appendix B-3: Generating Statspack Reports (continued)

```
End of Report ( sp_1_2.lst )
```

```
SQL>exit;
```

2. Examine the report that you created. You can use the `less` utility, which uses `vi` navigation commands, or the `gedit` graphic text viewer. What are your conclusions?
  - a) Invoke a viewer. (`gedit` is shown.) Navigation in `gedit` is similar to other graphic-based editors. To find a section of the report, use Search > Find and the report section name.

```
$ cd $HOME/labs
$ gedit sp_1_2.lst
```

- b) Scroll to the Top 5 Timed Events section. “Buffer busy waits” is among Top 5 Timed Events.
- c) View Time Model System Stats. The top entry is “sql execute elapsed time.”
- d) Looking at the Buffer Wait Statistics section, you can see that most of the waits are due to the data block category.
- e) Looking at the “Segments by Buffer Busy Waits” section, SPCT is shown as the culprit segment in the TBSSPC tablespace.
- f) All this leads you to think that there is an issue with the SPCT table. Looking at the tablespace definition, you discover that the table was defined without Automatic Segment Space Management.

```
$ sqlplus spc/spc
...
SQL> SELECT segment_space_management
2     FROM dba_tablespaces
3     WHERE tablespace_name=(select tablespace_name
4                             from user_tables);

SEGMENT
-----
MANUAL

SQL> exit;
```

3. Fix the problem by implementing the recommendations from the previous step. You decide to re-create the table by using Automatic Segment Space Management. This script is available as `sol_21_03_03.sql` in the solutions directory.

```
$ sqlplus / as sysdba
```

## Practice for Appendix B-3: Generating Statspack Reports (continued)

```
...  
  
SQL> @../solutions/sol_21_03_03  
SQL>  
SQL> drop tablespace tbsspc including contents and datafiles;  
  
Tablespace dropped.  
  
SQL>  
SQL> CREATE SMALLFILE TABLESPACE "TBSSPC"  
2 DATAFILE 'tbsspc1.dbf' SIZE 50M  
3 LOGGING  
4 EXTENT MANAGEMENT LOCAL  
5 SEGMENT SPACE MANAGEMENT AUTO;  
  
Tablespace created.  
  
SQL>  
SQL> connect spc/spc  
Connected.  
SQL>  
SQL> create table spct(id number, name varchar2(2000))  
tablespace tbsspc;  
  
Table created.  
  
SQL>  
SQL> exec DBMS_STATS.GATHER_TABLE_STATS(-  
> ownname=>'SPC', tabname=>'SPCT',-  
> estimate_percent=>DBMS_STATS.AUTO_SAMPLE_SIZE);  
  
PL/SQL procedure successfully completed.  
  
SQL>  
SQL> exit;  
...  
$
```

4. Regenerate the same workload, and verify that the problem disappears.
  - a) Take a new snapshot like you did in the previous practice. Run the same workload again and take another snapshot. Using the `$HOME/solutions/sol_21_03_04a.sh` script, you can create a snapshot, run the workload, and take a second snapshot script. Note the snap ID values. The script takes two minutes to complete.

```
$ $HOME/solutions/sol_21_03_04a.sh
```

- b) Generate the corresponding Statspack report using the snap ID values generated in the previous step.

## Practice for Appendix B-3: Generating Statspack Reports (continued)

```
$ sqlplus perfstat/perfstat

SQL> @?/rdbms/admin/spreport

...
Listing all Completed Snapshots
```

Instance Comment	DB Name	Snap Id	Snap Started	Snap Level
orcl	ORCL	1	16 Mar 2010 09:54	7
		2	16 Mar 2010 09:57	7
		3	16 Mar 2010 10:10	7
		4	16 Mar 2010 10:12	7

```
Specify the Begin and End Snapshot Ids
~~~~~
Enter value for begin_snap: 3
Begin Snapshot Id specified: 3

Enter value for end_snap: 4
End Snapshot Id specified: 4

Specify the Report Name
~~~~~
The default report file name is sp_3_4. To use this name,
press <return> to continue, otherwise enter an alternative.

Enter value for report_name: sp_3_4.lst
```

- c) Look at the generated report by using your preferred text browser. You can see that the number of buffer busy waits has been reduced.

