

Oracle Database 12c: SQL Tuning for Developers

Activity Guide

D79995GC10
Edition 1.0
October 2013
D84111

ORACLE®

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Disclaimer

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

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

Restricted Rights Notice

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

U.S. GOVERNMENT RIGHTS

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

Trademark Notice

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Author

Dimpi Rani Sarmah

Technical Contributors and Reviewers

Diganta Choudhury, Laszlo Czinkoczki, Joel Goodman, Nancy Greenberg, Naoki Kato, Sean Kim, Bryan Roberts, Swarnapriya Shridhar, Raza Siddiqui, Branislav Valny

This book was published using: *Oracle*tutor****

Table of Contents

| | |
|--|-------------|
| Practices for Lesson 1: Course Introduction | 1-1 |
| Practices for Lesson 1: Overview..... | 1-3 |
| Practices for Lesson 2: Introduction to SQL Tuning..... | 2-1 |
| Practices for Lesson 2: Overview..... | 2-2 |
| Practice 2: Using SQL Developer | 2-3 |
| Practices for Lesson 3: Using Application Tracing Tools..... | 3-1 |
| Practices for Lesson 3: Overview..... | 3-3 |
| Practice 3: Tracing Applications..... | 3-4 |
| Practices for Lesson 4: Using Basic Techniques | 4-1 |
| Practices for Lesson 4: Overview..... | 4-2 |
| Practice 4: Avoiding Common Mistakes | 4-3 |
| Practices for Lesson 5: Optimizer Fundamentals..... | 5-1 |
| Practices for Lesson 5: Overview..... | 5-2 |
| Practice 5-1: Understanding Optimizer Decisions (Optional) | 5-3 |
| Practices for Lesson 6: Generating and Displaying Execution Plans | 6-1 |
| Practices for Lesson 6: Overview..... | 6-2 |
| Practice 6-1: Extracting an Execution Plan by Using SQL Developer | 6-3 |
| Practice 6-2: Extracting Execution Plans..... | 6-11 |
| Practices for Lesson 7: Interpreting Execution Plans and Enhancements | 7-1 |
| Practices for Lesson 7: Overview..... | 7-2 |
| Practice 7-1: Using Dynamic Plans | 7-3 |
| Practices for Lesson 8: Optimizer: Table and Index Access Paths | 8-1 |
| Practices for Lesson 8: Overview..... | 8-2 |
| Practice 8-1: Using Different Access Paths | 8-3 |
| Practices for Lesson 9: Optimizer: Join Operators..... | 9-1 |
| Practices for Lesson 9: Overview..... | 9-2 |
| Practice 9: Using Join Paths..... | 9-3 |
| Practices for Lesson 10: Other Optimizer Operators..... | 10-1 |
| Practices for Lesson 10: Overview..... | 10-2 |
| Practice 10-1: Using the Result Cache..... | 10-3 |
| Practice 10-2: Using Other Access Paths (Optional)..... | 10-22 |
| Practices for Lesson 11: Introduction to Optimizer Statistics Concepts..... | 11-1 |
| Practices for Lesson 11: Overview..... | 11-2 |
| Practice 11-1: Index Clustering Factor | 11-3 |
| Practice 11-2: Creating Expression Statistics | 11-10 |
| Practice 11-3: Enabling Automatic Statistics Gathering (Optional)..... | 11-15 |
| Practice 11- 4: Using System Statistics (Optional) | 11-38 |
| Practices for Lesson 12: Using Bind Variables..... | 12-1 |
| Practices for Lesson 12: Overview..... | 12-2 |
| Practice 12-1: Understanding Adaptive Cursor Sharing | 12-3 |
| Practice 12-2: Understanding CURSOR_SHARING (Optional) | 12-22 |
| Practices for Lesson 13: SQL Plan Management..... | 13-1 |
| Practices for Lesson 13: Overview..... | 13-2 |
| Practice 13-1: Using SQL Plan Management (SPM)..... | 13-3 |

| | |
|--|-------------|
| Workshop 1..... | 14-1 |
| Workshop 1: Overview | 14-2 |
| Workshop 1: Enhancing the Performance of a SQL Query Statement..... | 14-3 |
| Workshop 2..... | 15-1 |
| Workshop 2: Overview | 15-2 |
| Workshop 2: Reviewing the Execution Steps of the SQL Statement..... | 15-3 |
| Workshop 3..... | 16-1 |
| Workshop 3 | 16-2 |
| Workshop 3: Learn to Tune Sort Operation Using an Index in the ORDER BY Clauses..... | 16-3 |
| Workshop 4..... | 17-1 |
| Workshop 4: Overview | 17-2 |
| Workshop 4: Identifying and Tuning a Poorly Written SQL Statement | 17-3 |
| Workshop 5..... | 18-1 |
| Workshop 5: Overview | 18-2 |
| Workshop 5: Effects of Changing the Column Order in a Composite Index..... | 18-3 |
| Workshop 6..... | 19-1 |
| Workshop 6: Overview | 19-2 |
| Workshop-6: Using Information in the 10053 File to Tune a SQL Statement..... | 19-3 |
| Workshop 7..... | 20-1 |
| Workshop 7: Overview | 20-2 |
| Workshop-7: Understanding the Optimizer's Decision | 20-3 |
| Workshop 8..... | 21-1 |
| Workshop 8: Overview | 21-2 |
| Workshop 8: Tuning Strategy | 21-3 |
| Workshop 9..... | 22-1 |
| Workshop 9: Overview | 22-2 |
| Workshop 9: Using SQL Plan Baseline to Manage a Better Execution Plan | 22-3 |

Practices for Lesson 1: Course Introduction

Chapter 1

Practices for Lesson 1: Overview

Practices Overview

There are no practices for lesson 1.

Practices for Lesson 2: Introduction to SQL Tuning

Chapter 2

Practices for Lesson 2: Overview

Practices Overview

This practice covers using SQL Developer.

Practice 2: Using SQL Developer

Overview

In this practice, you get acquainted with SQL Developer functions.

Tasks

1. Start Oracle SQL Developer.

Double-click the Oracle SQL Developer icon on your desktop.



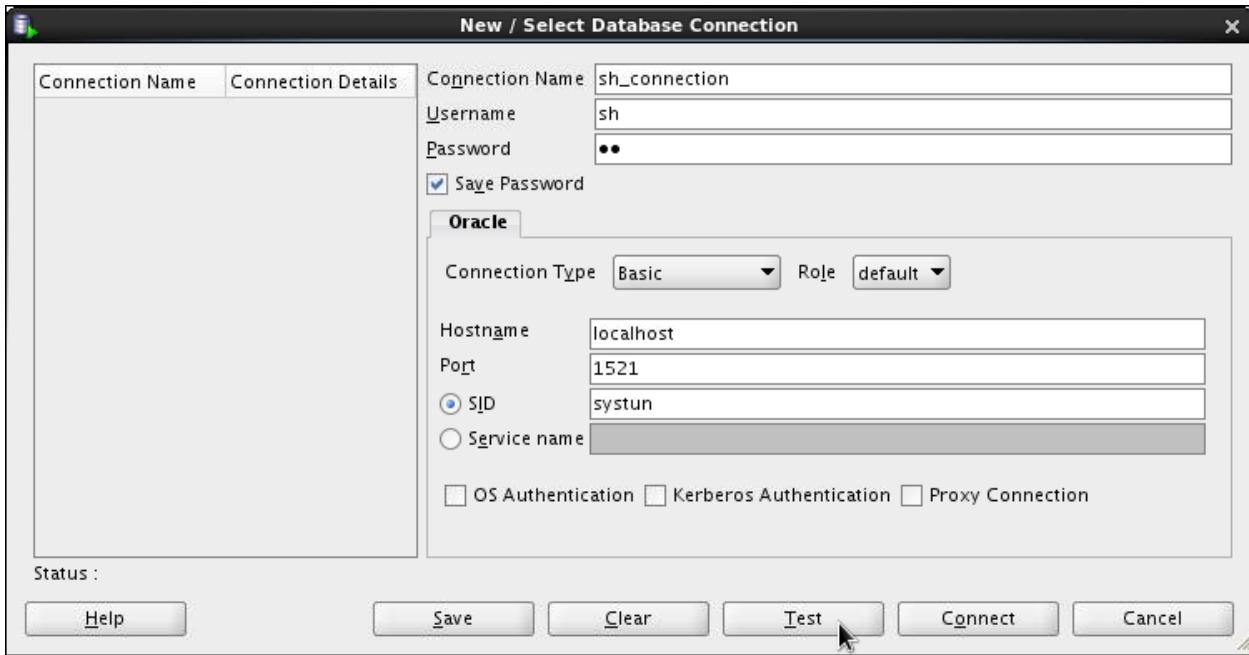
2. Create a database connection to the SH schema by using the following information:

| | |
|-----------------|---------------|
| Connection Name | sh_connection |
| Username | sh |
| Password | sh |
| Hostname | localhost |
| Port | 1521 |
| SID | systun |

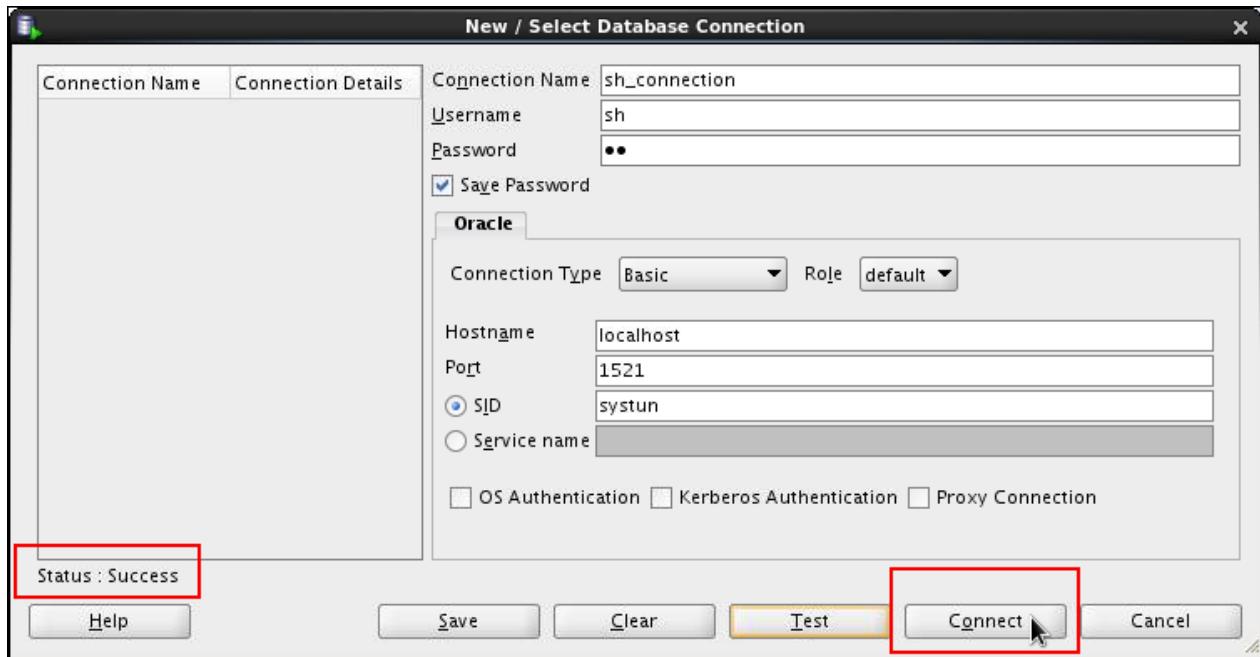
Right-click the Connections icon on the Connections tabbed page, and then select **New Connection** from the shortcut menu. The **New / Select Database Connection Window** appears. Use the preceding information to create the new database connection.

Note: To display the properties of the newly created connection, right-click the connection name, and then select Properties from the shortcut menu. Enter the username, password, host name, and service name with the appropriate information, as provided in the table. Select the “**Save Password**” check box.

The following is a sample of the newly created database connection for the SH schema using a local connection:

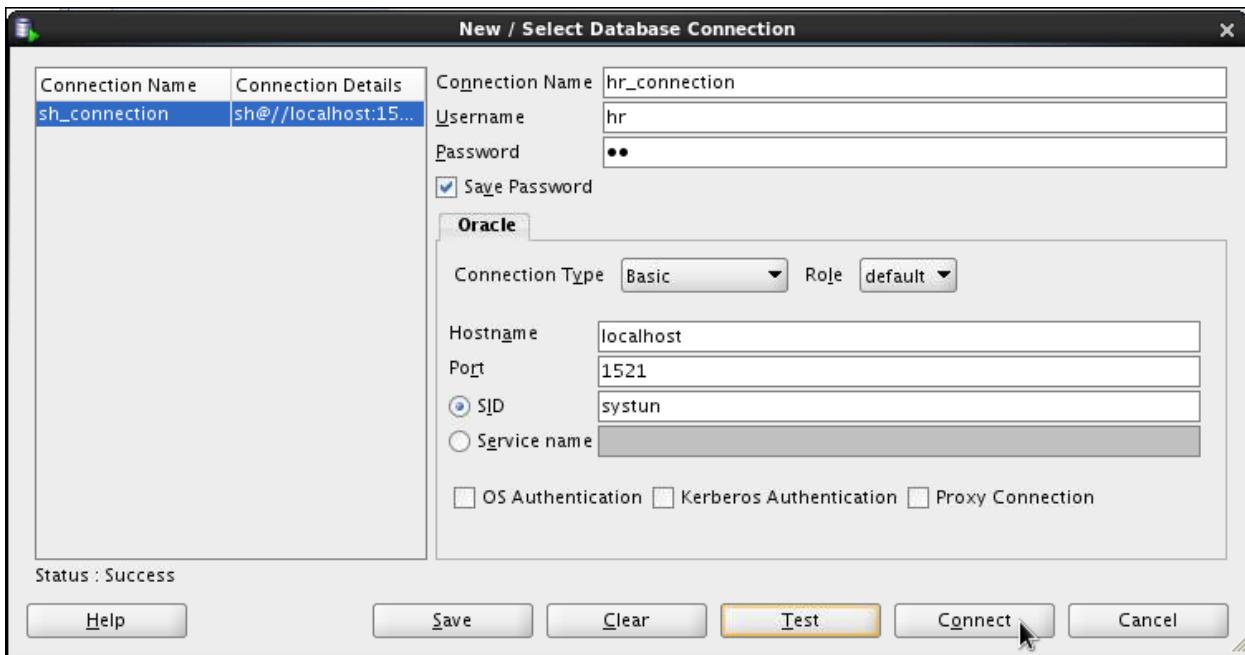


3. Test the new connection. If the Status is Success, save the connection and then connect to the database by using this new connection.

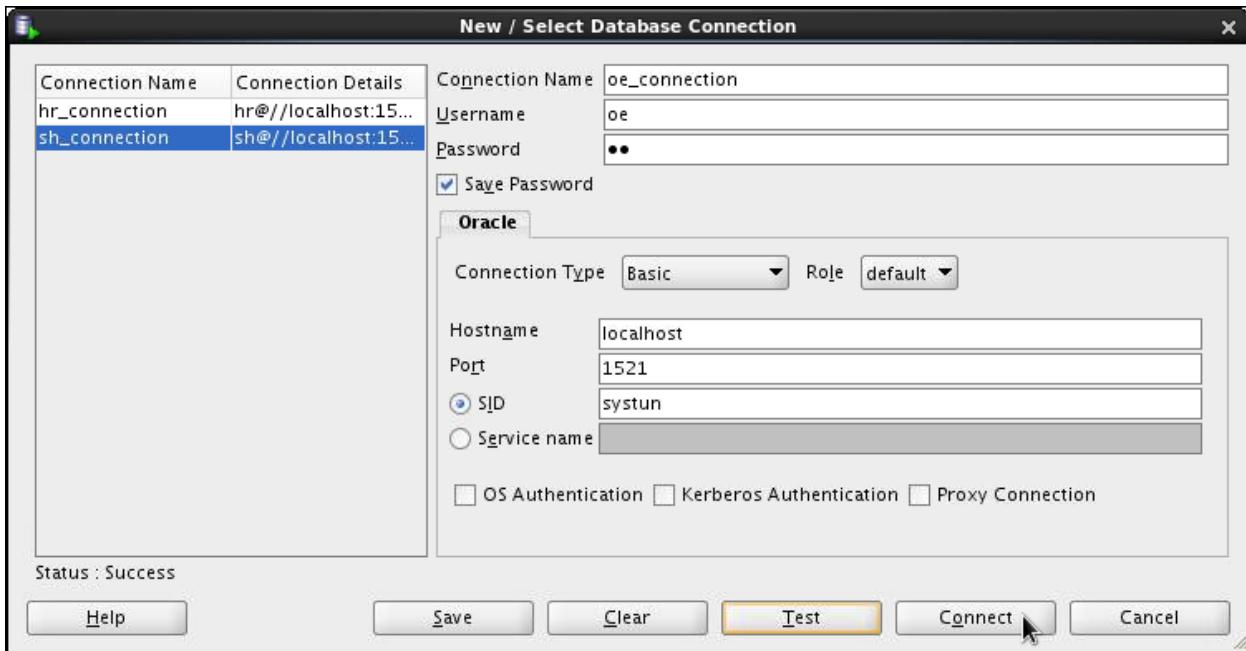


4. Create a new database connection named hr_connection.
 - a. Right-click sh_connection in the Object Navigation tree, and select the Properties menu option.

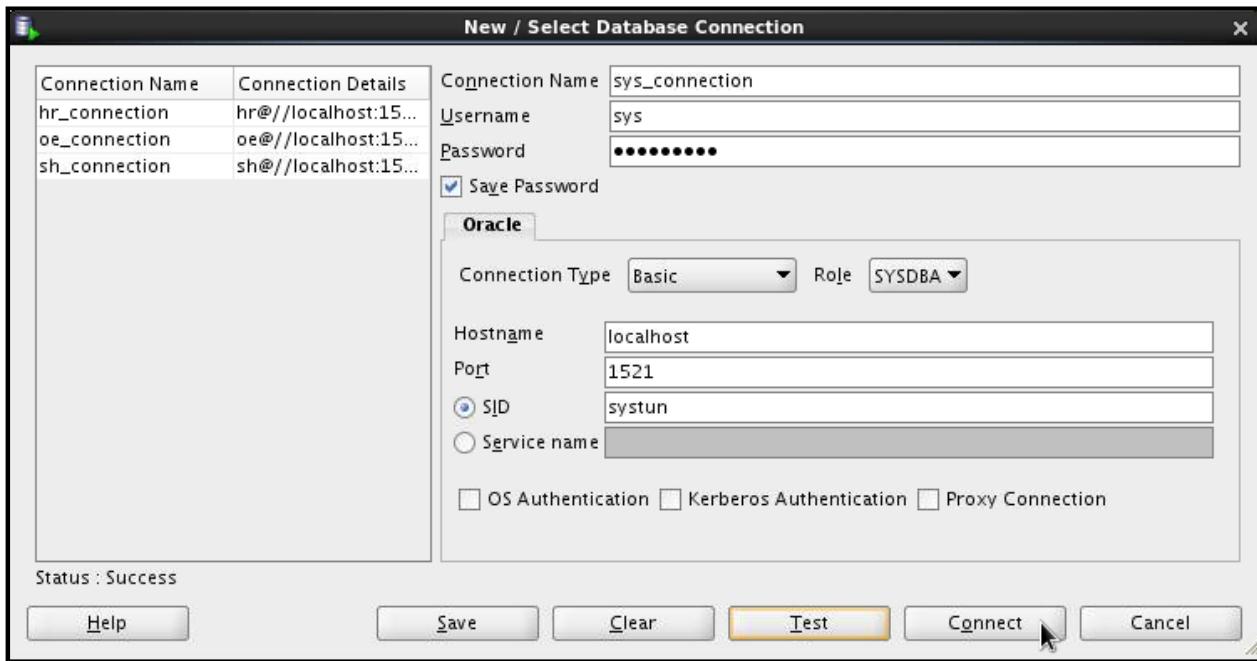
- b. Enter `hr_connection` as the connection name and `hr` as the username and password, select Save Password, and then click Test to test the new connection.
- c. Click Connect to connect to the database.



5. Repeat steps 2 and 3 to create and test a new database connection named `oe_connection`. Enter `oe` as the database connection username and password.

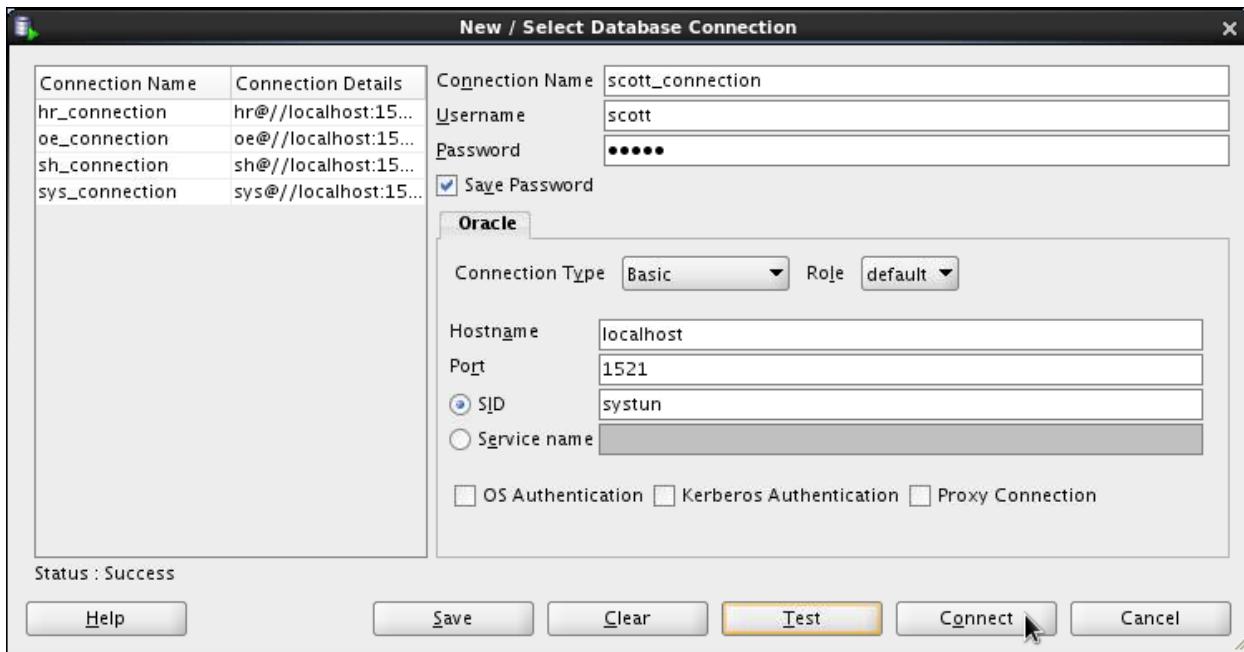


6. Repeat steps 2 and 3 to create and test a new database connection named `sys_connection`. Enter `sys` in the Username field and `oracle_4U` in the Password field. From the Role drop-down menu, select `SYSDBA` as the role.



7. Create a new database connection named `scott_connection`.
- Right-click `sys_connection` in the Object Navigation tree, and select the Properties menu option.
 - Enter `scott_connection` as the connection name and `scott` as the username and `tiger` as password, select Save Password, and then click Test to test the new connection.

- c. Click Connect to connect to the database.



Browsing the HR, SH, and OE Schema Tables

8. Browse the structure of the EMPLOYEES table in the HR schema.
 - a. Expand the hr_connection connection by clicking the plus sign.
 - b. Expand Tables by clicking the plus sign.
 - c. Display the structure of the EMPLOYEES table. Click the EMPLOYEES table. The Columns tab displays the columns in the EMPLOYEES table as follows:

| COLUMN_NAME | DATA_TYPE | NULLABLE | DATA_DEFAULT | COLUMN_ID | COMMENTS |
|------------------|-------------------|----------|--------------|-----------|--------------------------------------|
| 1 EMPLOYEE_ID | NUMBER(6,0) | No | (null) | 1 | 1 Primary key of employees table. |
| 2 FIRST_NAME | VARCHAR2(20 BYTE) | Yes | (null) | 2 | 2 First name of the employee. A not |
| 3 LAST_NAME | VARCHAR2(25 BYTE) | No | (null) | 3 | 3 Last name of the employee. A not |
| 4 EMAIL | VARCHAR2(25 BYTE) | No | (null) | 4 | 4 Email id of the employee |
| 5 PHONE_NUMBER | VARCHAR2(20 BYTE) | Yes | (null) | 5 | 5 Phone number of the employee; inc |
| 6 HIRE_DATE | DATE | No | (null) | 6 | 6 Date when the employee started on |
| 7 JOB_ID | VARCHAR2(10 BYTE) | No | (null) | 7 | 7 Current job of the employee; fore |
| 8 SALARY | NUMBER(8,2) | Yes | (null) | 8 | 8 Monthly salary of the employee. M |
| 9 COMMISSION_PCT | NUMBER(2,2) | Yes | (null) | 9 | 9 Commission percentage of the empl |
| 10 MANAGER_ID | NUMBER(6,0) | Yes | (null) | 10 | 10 Manager id of the employee; has s |
| 11 DEPARTMENT_ID | NUMBER(4,0) | Yes | (null) | 11 | 11 Department id where employee work |

9. Browse the EMPLOYEES table and display its data.
 - a. To display the employee data, click the Data tab. The EMPLOYEES table data is displayed as follows:

| | EMPLOYEE_ID | FIRST_NAME | LAST_NAME | EMAIL | PHONE_NUMBER | HIRE_DATE | JOB_ID | SALARY |
|----|-------------|------------|-----------|----------|--------------|-----------|------------|--------|
| 1 | 100 | Steven | King | SKING | 515.123.4567 | 17-JUN-03 | AD_PRES | 24000 |
| 2 | 101 | Neena | Kochhar | NKOCHHAR | 515.123.4568 | 21-SEP-05 | AD_VP | 17000 |
| 3 | 102 | Lex | De Haan | LDEHAAN | 515.123.4569 | 13-JAN-01 | AD_VP | 17000 |
| 4 | 103 | Alexander | Hunold | AHUNOLD | 590.423.4567 | 03-JAN-06 | IT_PROG | 9000 |
| 5 | 104 | Bruce | Ernst | BERNST | 590.423.4568 | 21-MAY-07 | IT_PROG | 6000 |
| 6 | 105 | David | Austin | DAUSTIN | 590.423.4569 | 25-JUN-05 | IT_PROG | 4800 |
| 7 | 106 | Valli | Pataballa | VPATABAL | 590.423.4560 | 05-FEB-06 | IT_PROG | 4800 |
| 8 | 107 | Diana | Lorentz | DLORENTZ | 590.423.5567 | 07-FEB-07 | IT_PROG | 4200 |
| 9 | 108 | Nancy | Greenberg | NGREENBE | 515.124.4569 | 17-AUG-02 | FI_MGR | 12008 |
| 10 | 109 | Daniel | Faviet | DFAVIET | 515.124.4169 | 16-AUG-02 | FI_ACCOUNT | 9000 |
| 11 | 110 | John | Chen | JCHEN | 515.124.4269 | 28-SEP-05 | FI_ACCOUNT | 8200 |
| 12 | 111 | Ismael | Sciarra | ISCIARRA | 515.124.4369 | 30-SEP-05 | FI_ACCOUNT | 7700 |

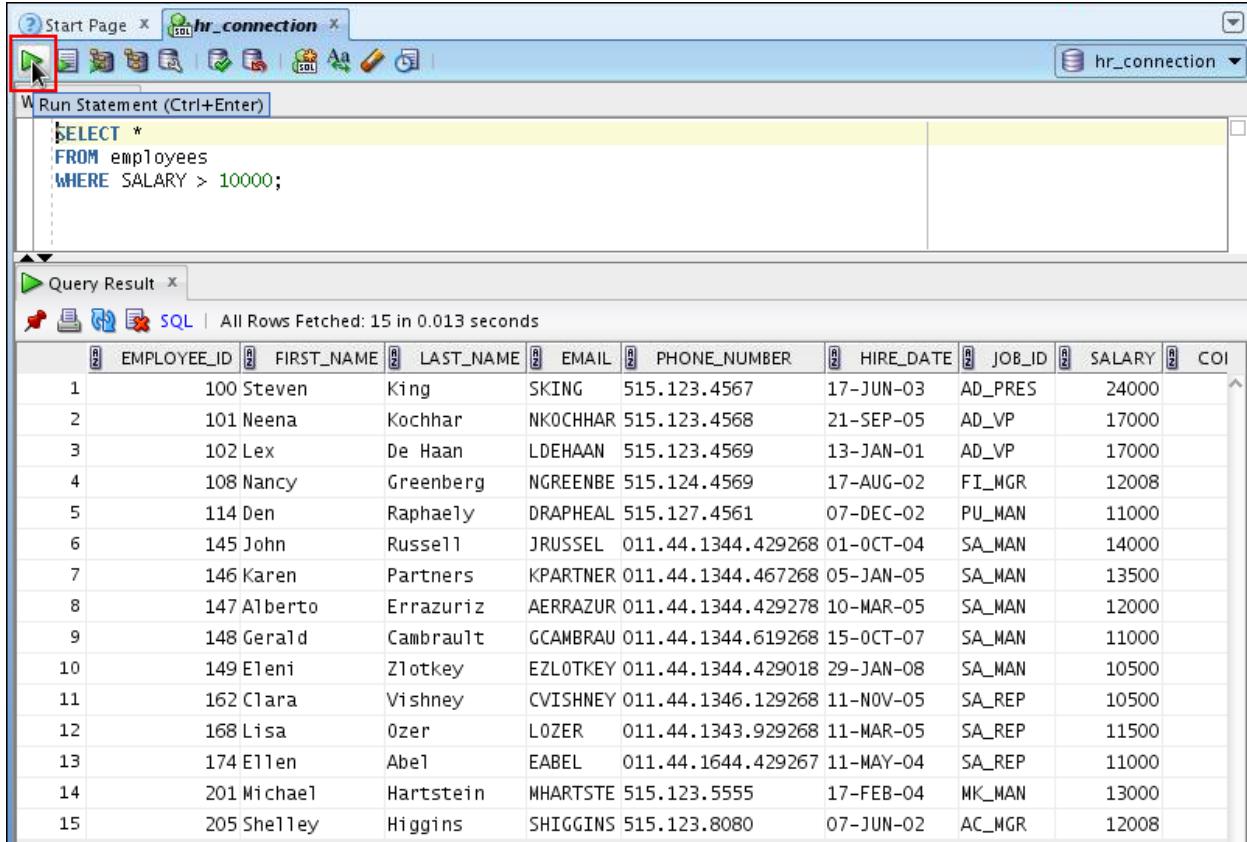
...

10. Use the SQL Worksheet to select the last names and salaries of all employees whose annual salary is greater than \$10,000. Use both the Execute Statement (F9) and the Run Script (F5) icons to execute the SELECT statement. Review the results of both methods of executing the SELECT statements on the appropriate tabs.

Display the SQL Worksheet by using any of the following two methods: Select Tools > SQL Worksheet, or click the Open SQL Worksheet icon. The Select Connection window appears. Select hr_connection. Enter the following statement in the SQL Worksheet:

```
SELECT *
FROM employees
WHERE SALARY > 10000;
```

- 1) Execute by pressing Ctrl + Enter or by clicking the Run Statement icon .



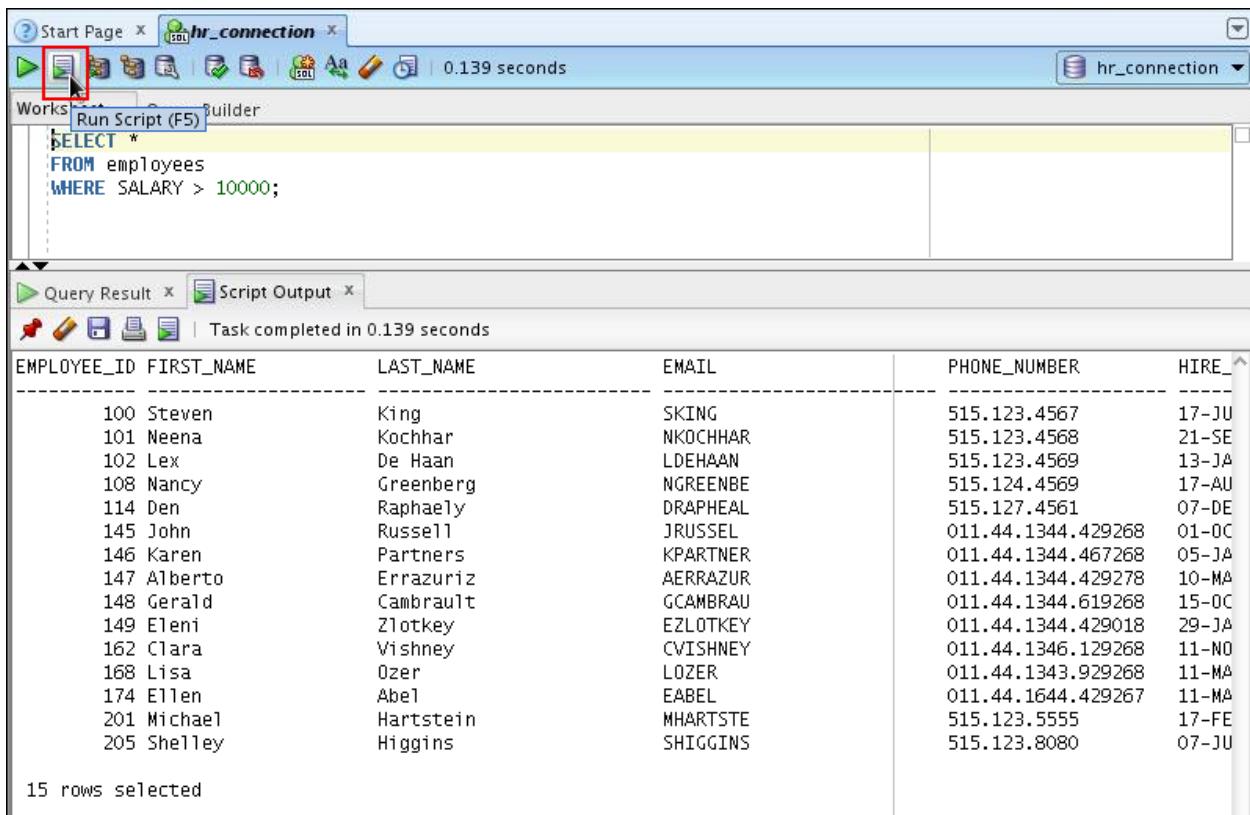
The screenshot shows the Oracle SQL Developer interface. The top bar has tabs for 'Start Page' and 'hr_connection'. Below the bar is a toolbar with various icons. A red box highlights the 'Run Statement (Ctrl+Enter)' icon, which is a green triangle pointing right. The main area has two panes. The left pane, titled 'Run Statement (Ctrl+Enter)', contains the SQL query:

```
SELECT *
FROM employees
WHERE SALARY > 10000;
```

. The right pane, titled 'Query Result', displays the results of the query. The results table has 15 rows and columns for Employee ID, First Name, Last Name, Email, Phone Number, Hire Date, Job ID, Salary, and CO. The data is as follows:

| | EMPLOYEE_ID | FIRST_NAME | LAST_NAME | EMAIL | PHONE_NUMBER | HIRE_DATE | JOB_ID | SALARY | CO |
|----|-------------|------------|-----------|----------|--------------------|-----------|---------|--------|----|
| 1 | 100 | Steven | King | SKING | 515.123.4567 | 17-JUN-03 | AD_PRES | 24000 | |
| 2 | 101 | Neena | Kochhar | NKOCHHAR | 515.123.4568 | 21-SEP-05 | AD_VP | 17000 | |
| 3 | 102 | Lex | De Haan | LDEHAAN | 515.123.4569 | 13-JAN-01 | AD_VP | 17000 | |
| 4 | 108 | Nancy | Greenberg | NGREENBE | 515.124.4569 | 17-AUG-02 | FI_MGR | 12008 | |
| 5 | 114 | Den | Raphaely | DRAPHEAL | 515.127.4561 | 07-DEC-02 | PU_MAN | 11000 | |
| 6 | 145 | John | Russell | JRUSSEL | 011.44.1344.429268 | 01-OCT-04 | SA_MAN | 14000 | |
| 7 | 146 | Karen | Partners | KPARTNER | 011.44.1344.467268 | 05-JAN-05 | SA_MAN | 13500 | |
| 8 | 147 | Alberto | Errazuriz | AERRAZUR | 011.44.1344.429278 | 10-MAR-05 | SA_MAN | 12000 | |
| 9 | 148 | Gerald | Cambrault | GCAMBRAU | 011.44.1344.619268 | 15-OCT-07 | SA_MAN | 11000 | |
| 10 | 149 | Eleni | Zlotkey | EZLOTKEY | 011.44.1344.429018 | 29-JAN-08 | SA_MAN | 10500 | |
| 11 | 162 | Clara | Vishney | CVISHNEY | 011.44.1346.129268 | 11-NOV-05 | SA REP | 10500 | |
| 12 | 168 | Lisa | Ozer | LOZER | 011.44.1343.929268 | 11-MAR-05 | SA REP | 11500 | |
| 13 | 174 | Ellen | Abel | EABEL | 011.44.1644.429267 | 11-MAY-04 | SA REP | 11000 | |
| 14 | 201 | Michael | Hartstein | MHARTSTE | 515.123.5555 | 17-FEB-04 | MKT_MAN | 13000 | |
| 15 | 205 | Shelley | Higgins | SHIGGINS | 515.123.8080 | 07-JUN-02 | AC_MGR | 12008 | |

- 2) Execute by pressing the F5 key or by clicking the Execute Script icon .



The screenshot shows the Oracle SQL Developer interface. In the top-left corner of the main window, there is a red box highlighting the 'Run Script (F5)' button, which corresponds to the Execute Script icon mentioned in the task. The SQL editor tab is active, displaying the following SQL query:

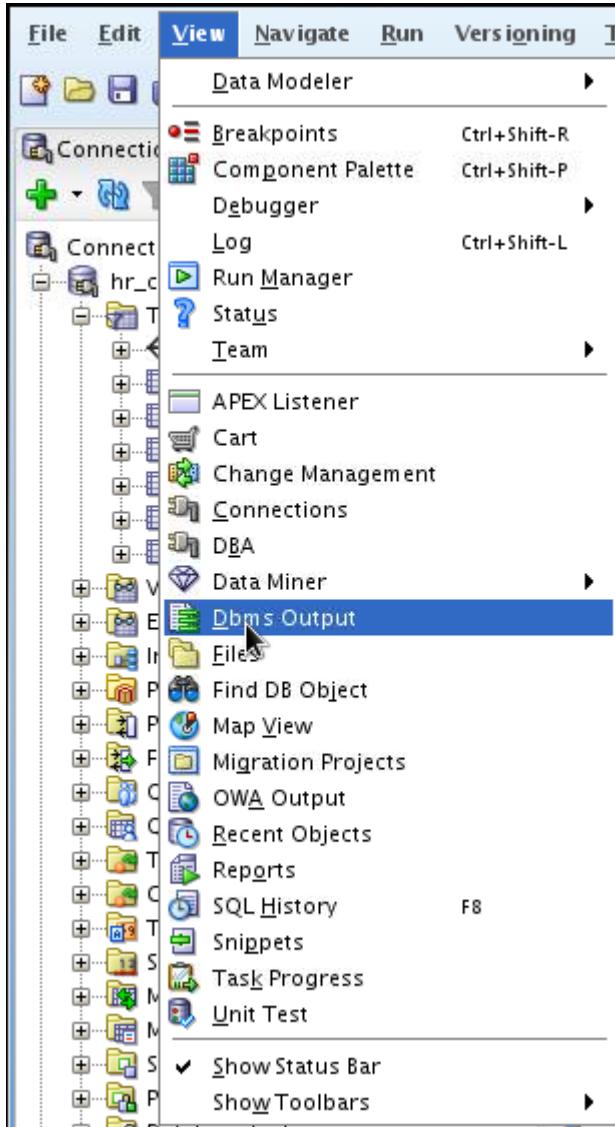
```
SELECT *
FROM employees
WHERE SALARY > 10000;
```

Below the editor, the 'Query Result' tab is selected, showing the results of the executed query. The results are presented in a table with the following columns: EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMAIL, PHONE_NUMBER, and HIRE_DATE. The data shows 15 rows of employee information where salary is greater than 10000.

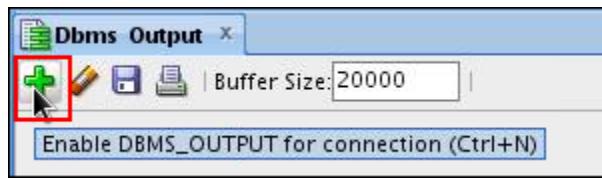
| EMPLOYEE_ID | FIRST_NAME | LAST_NAME | EMAIL | PHONE_NUMBER | HIRE_DATE |
|-------------|------------|-----------|----------|--------------------|-----------|
| 100 | Steven | King | SKING | 515.123.4567 | 17-JU |
| 101 | Neena | Kochhar | NKOCHHAR | 515.123.4568 | 21-SE |
| 102 | Lex | De Haan | LDEHAAN | 515.123.4569 | 13-JA |
| 108 | Nancy | Greenberg | NGREENBE | 515.124.4569 | 17-AU |
| 114 | Den | Raphaely | DRAPHEAL | 515.127.4561 | 07-DE |
| 145 | John | Russell | JRUSSEL | 011.44.1344.429268 | 01-OC |
| 146 | Karen | Partners | KPARTNER | 011.44.1344.467268 | 05-JA |
| 147 | Alberto | Errazuriz | AERRAZUR | 011.44.1344.429278 | 10-MA |
| 148 | Gerald | Cambrault | GCAMBRAU | 011.44.1344.619268 | 15-OC |
| 149 | Eleni | Zlotkey | EZLOTKEY | 011.44.1344.429018 | 29-JA |
| 162 | Clara | Vishney | CVISHNEY | 011.44.1346.129268 | 11-NO |
| 168 | Lisa | Ozer | LOZER | 011.44.1343.929268 | 11-MA |
| 174 | Ellen | Abel | EABEL | 011.44.1644.429267 | 11-MA |
| 201 | Michael | Hartstein | MHARTSTE | 515.123.5555 | 17-FE |
| 205 | Shelley | Higgins | SHIGGINS | 515.123.8080 | 07-JU |

15 rows selected

11. In the same SQL Worksheet, create and execute a simple anonymous block that outputs "Hello World."
- Enable SET SERVEROUTPUT ON to display the output of the DBMS_OUTPUT package statements. Select View > Dbms Output.



- Click the green plus sign, as shown below, to enable DBMS_OUTPUT.

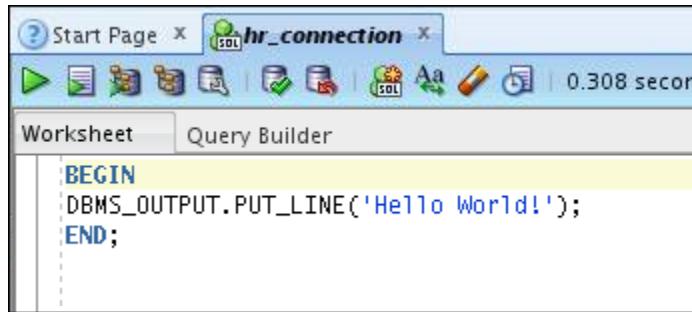


- c. In the dialog box that appears, select hr_connection and click OK.



- d. Use the SQL Worksheet area to enter the code for your anonymous block.

```
BEGIN  
DBMS_OUTPUT.PUT_LINE('Hello World!');  
END;
```



- e. Click the Execute icon (Ctrl + Enter) to run the anonymous block.
The Script Output tab displays the output of the anonymous block as follows:

The screenshot shows the Oracle SQL Developer interface. In the top-left corner, there's a toolbar with various icons. The main window has tabs for 'Start Page' and 'hr_connection'. The 'Worksheet' tab is active, showing the following anonymous PL/SQL block:

```
BEGIN  
DBMS_OUTPUT.PUT_LINE('Hello World!');  
END;
```

Below the Worksheet is the 'Script Output' tab, which displays the message: "anonymous block completed". At the bottom of the interface is the 'Dbms Output' tab, which shows the output "Hello World!". A tooltip for the 'Run Statement (Ctrl+Enter)' button is visible above the Worksheet tab.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Error : You are not a Valid Partner use only

Practices for Lesson 3: Using Application Tracing Tools

Chapter 3

Practices for Lesson 3: Overview

Practices Overview

This practice covers the following topics:

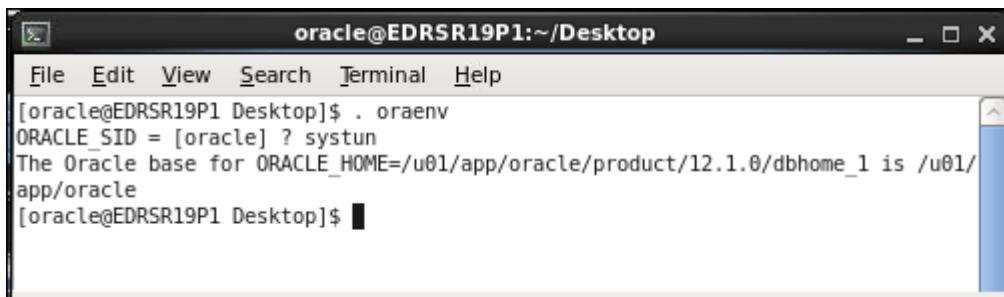
- Creating a service
- Tracing your application by using services
- Interpreting trace information by using `trcseess` and `TKPROF`

Practice 3: Tracing Applications

Overview

In this practice, you define a service and use it to generate traces. You then interpret generated trace files. You can find all needed script files for this practice in your home/oracle/labs/solutions/Application_Tracing directory.

Note: Before executing the scripts, set your environment by using the following code:



```
oracle@EDRSR19P1:~/Desktop
File Edit View Search Terminal Help
[oracle@EDRSR19P1 Desktop]$ . oraenv
ORACLE SID = [oracle] ? systun
The Oracle base for ORACLE_HOME=/u01/app/oracle/product/12.1.0/dbhome_1 is /u01/app/oracle
[oracle@EDRSR19P1 Desktop]$
```

Tasks

1. Your environment has been initialized with the `at_setup.sql` script before the class began. The script is listed below:

```
set echo on

drop user trace cascade;

create user trace identified by trace default tablespace users
temporary tablespace temp;

grant connect, resource, dba to trace;

drop tablespace tracetbs including contents and datafiles;

drop tablespace tracetbs3 including contents and datafiles;

create tablespace tracetbs
datafile '/u01/app/oracle/oradata/systun/tracetbs.dbf' size 100m
extent management local uniform size 40k;

create tablespace tracetbs3
datafile '/u01/app/oracle/oradata/systun/tracetbs3.dbf' size 100m
extent management local uniform size 10m;

connect trace/trace

drop table sales purge;

create table sales as select * from sh.sales;
```

```
drop table sales2 purge;

create table sales2 tablespace tracetbs as select * from sh.sales
where 1=2;

drop table sales3 purge;

create table sales3 tablespace tracetbs3 as select * from sh.sales
where 1=2;

exit;
```

2. Before you can use a service in your application, you have to make sure that this service is available from the `tnsnames.ora` file you use to connect to your database. Modify the `tnsnames.ora` file to create a net service called `TRACESERV`. The `add_traceserv_tns.sh` script in the `/home/oracle/labs/solutions/Application_Tracing` directory helps you in this task, by replacing `node` in the `tnsnames.ora` file with your host name.

```
TRACESERV =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP) (HOST = node) (PORT = 1521))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = TRACESERV)
    )
  )
```

Note: In the following listing, `node` will be replaced with your host name.

```
$ cd /home/oracle/labs/solutions/Application_Tracing
$ ./add_traceserv_tns.sh
$ cat $ORACLE_HOME/network/admin/tnsnames.ora
```

```
[oracle@EDRSR19P1 Desktop]$ . oraenv
ORACLE_SID = [oracle] ? systun
The Oracle base for ORACLE_HOME=/u01/app/oracle/product/12.1.0/dbhome_1 is /u01/
app/oracle
[oracle@EDRSR19P1 Desktop]$ cd /home/oracle/labs/solutions/Application_Tracing
[oracle@EDRSR19P1 Application_Tracing]$ ./add_traceserv_tns.sh
[oracle@EDRSR19P1 Application_Tracing]$ cat $ORACLE_HOME/network/admin/tnsnames.
ora
# tnsnames.ora Network Configuration File: /u01/app/oracle/product/12.1.0/dbhome
_1/network/admin/tnsnames.ora
# Generated by Oracle configuration tools.

SYSTUN =
(DESCRIPTION =
  (ADDRESS = (PROTOCOL = TCP)(HOST = edRSr19p1.us.oracle.com)(PORT = 1521))
  (CONNECT_DATA =
    (SERVER = DEDICATED)
    (SERVICE_NAME = systun)
  )
)

EM12REP =
(DESCRIPTION =
  (ADDRESS = (PROTOCOL = TCP)(HOST = edRSr19p1.us.oracle.com)(PORT = 1521))
  (CONNECT_DATA =
    (SERVER = DEDICATED)
    (SERVICE_NAME = em12rep)
  )
)

TRACESERV =
(DESCRIPTION =
  (ADDRESS = (PROTOCOL = TCP)(HOST = EDRSR19P1)(PORT = 1521))
  (CONNECT_DATA =
    (SERVER = DEDICATED)
    (SERVICE_NAME = TRACESERV )
  )
)
```

3. You now need to declare the TRACESERV service in your database. So connect to your database instance as the SYS user by using a SQL*Plus session.

```
$ sqlplus / as sysdba

...
SQL>
```

4. In your SQL*Plus session, create a new service called TRACESERV with no domain name. Run the add_traceserv_db.sql file.

```
[oracle@EDRSR19P1 Application_Tracing]$ sqlplus / as sysdba
SQL*Plus: Release 12.1.0.1.0 Production on Wed Mar 13 23:14:21 2013
Copyright (c) 1982, 2012, Oracle. All rights reserved.

Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.1.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application Testing options

SQL> @add_traceserv_db
SQL>
SQL> select name from dba_services;

NAME
-----
SYS$BACKGROUND
SYS$USERS
systunXDB
systun

SQL>
SQL> exec DBMS_SERVICE.CREATE_SERVICE('TRACESERV', 'TRACESERV');

PL/SQL procedure successfully completed.

SQL>
SQL> select name from dba_services;

NAME
-----
SYS$BACKGROUND
SYS$USERS
TRACESERV
systunXDB
systun

SQL>
```

- From the same SQL*Plus session, start the TRACESERV service.

```
SQL> @start_traceserv
```

```
SQL> @start_traceserv
SQL> set echo on
SQL>
SQL> show parameter service_names

NAME          TYPE    VALUE
-----
service_names    string  systun
SQL>
SQL> exec DBMS_SERVICE.START_SERVICE('TRACESERV');

PL/SQL procedure successfully completed.

SQL>
SQL> show parameter service_names

NAME          TYPE    VALUE
-----
service_names    string  systun
SQL>
```

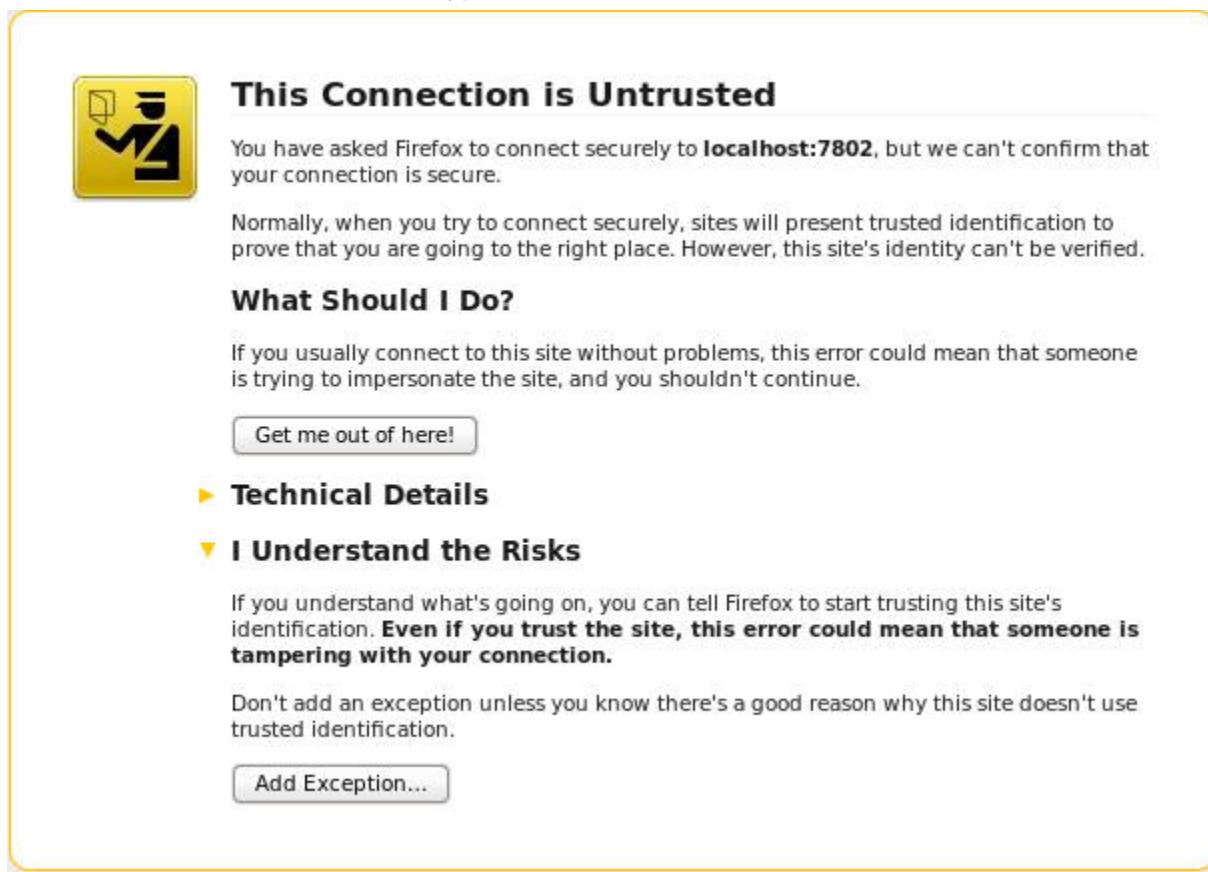
6. Exit from your SQL*Plus session.

```
SQL> exit;  
Disconnected ...  
  
$
```

7. Open a browser window and connect to Enterprise Manager Cloud Control as the SYSMAN user. Navigate to the Top Services page. The URL is <https://localhost:7802/em>.

Note: If you get an alert box saying “Secure Connection Failed,” then follow these steps:

Click the “I Understand the Risks” hyperlink.



The screenshot shows a Firefox browser window with a yellow warning dialog. The dialog title is "This Connection is Untrusted". It features a yellow icon of a person holding a briefcase. The main text says: "You have asked Firefox to connect securely to **localhost:7802**, but we can't confirm that your connection is secure." Below this, it states: "Normally, when you try to connect securely, sites will present trusted identification to prove that you are going to the right place. However, this site's identity can't be verified." There are two buttons at the bottom: "Get me out of here!" and "Add Exception...". A section titled "What Should I Do?" is visible above the buttons.

This Connection is Untrusted

You have asked Firefox to connect securely to **localhost:7802**, but we can't confirm that your connection is secure.

Normally, when you try to connect securely, sites will present trusted identification to prove that you are going to the right place. However, this site's identity can't be verified.

What Should I Do?

If you usually connect to this site without problems, this error could mean that someone is trying to impersonate the site, and you shouldn't continue.

Get me out of here! Add Exception...

► **Technical Details**

▼ **I Understand the Risks**

If you understand what's going on, you can tell Firefox to start trusting this site's identification. **Even if you trust the site, this error could mean that someone is tampering with your connection.**

Don't add an exception unless you know there's a good reason why this site doesn't use trusted identification.

Click the “Add Exception” button.

Click the “Get Certificate” button.



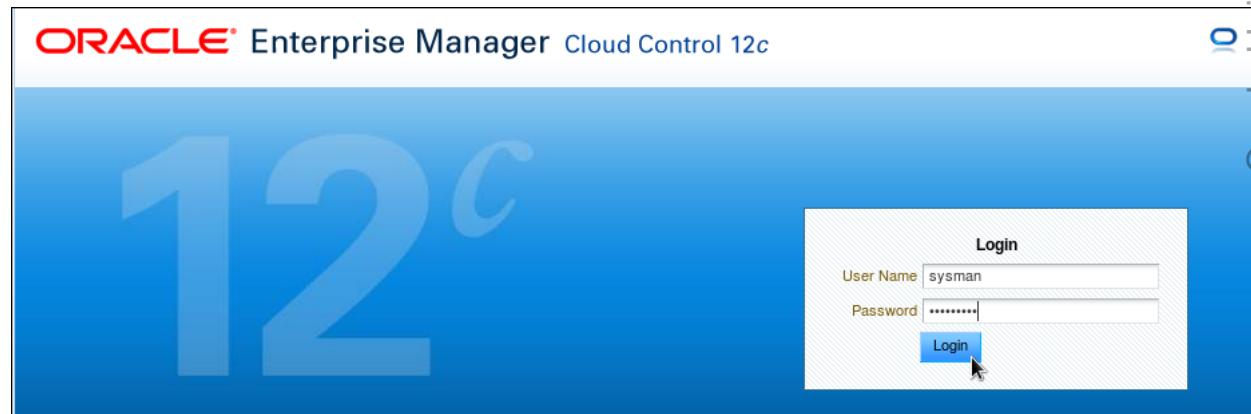
Confirm by clicking the “**Confirm Security Exception**” button.



- Log in to Enterprise Manager as the sysman user.

Username: sysman

Password: oracle_4U



- Click I Accept button.

ORACLE Enterprise Manager Cloud Control 12c

License Agreement

Clicking the "I Accept" button below confirms your agreement that you comply with each of the following statements:

Oracle Enterprise Manager provides central management of your entire Oracle environment. Note that usage of certain product features requires separate licenses for Oracle Enterprise Manager Management Packs. You can enable or disable access to Management Pack features to stay compliant with your license terms. This can be done for individual targets, or you can use batch update to enable or disable management packs for all targets of a specific type.

Management Pack features can be accessed through Oracle Enterprise Manager Cloud Control browser interface, EMCLI (Enterprise Manager Command-line Interface) as well as through APIs provided with Oracle Database software. Access with any of these methods requires appropriate Management Pack license. For detailed information on the functionality contained within these packs, please refer to the Oracle Database Licensing Information document, the Oracle Application Server Licensing Information document or the Oracle Enterprise Manager Licensing Information document.

- c. Go to the **Targets** tab and select **Databases**.
- d. For the **View** field, select the **Search List** radio button.
- e. Click **systun**.

ORACLE Enterprise Manager Cloud Control 12c

Enterprise Targets Favorites History

Databases

View Database Load Map Search List

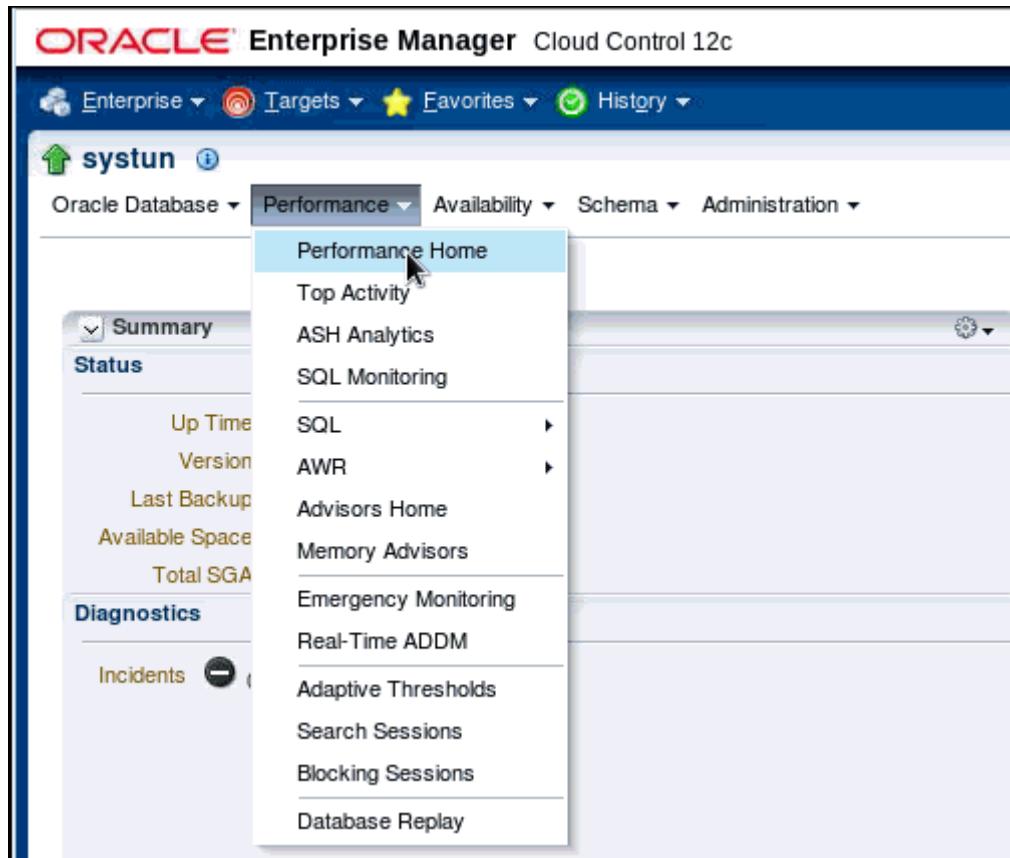
Search Advanced Search

|

| Select | Name | Status | Incidents | C |
|----------------------------------|--------|--------|------------|---|
| <input checked="" type="radio"/> | systun | | 0 1135 0 0 | |

TIP For an explanation of the icons and symbols used in this page, see the [Icon Key](#).

- f. On the Database Home page, select **Performance Home**.



- g. Log in to the Performance page as following:

Username: DEV

Password: DEV

Role: Normal

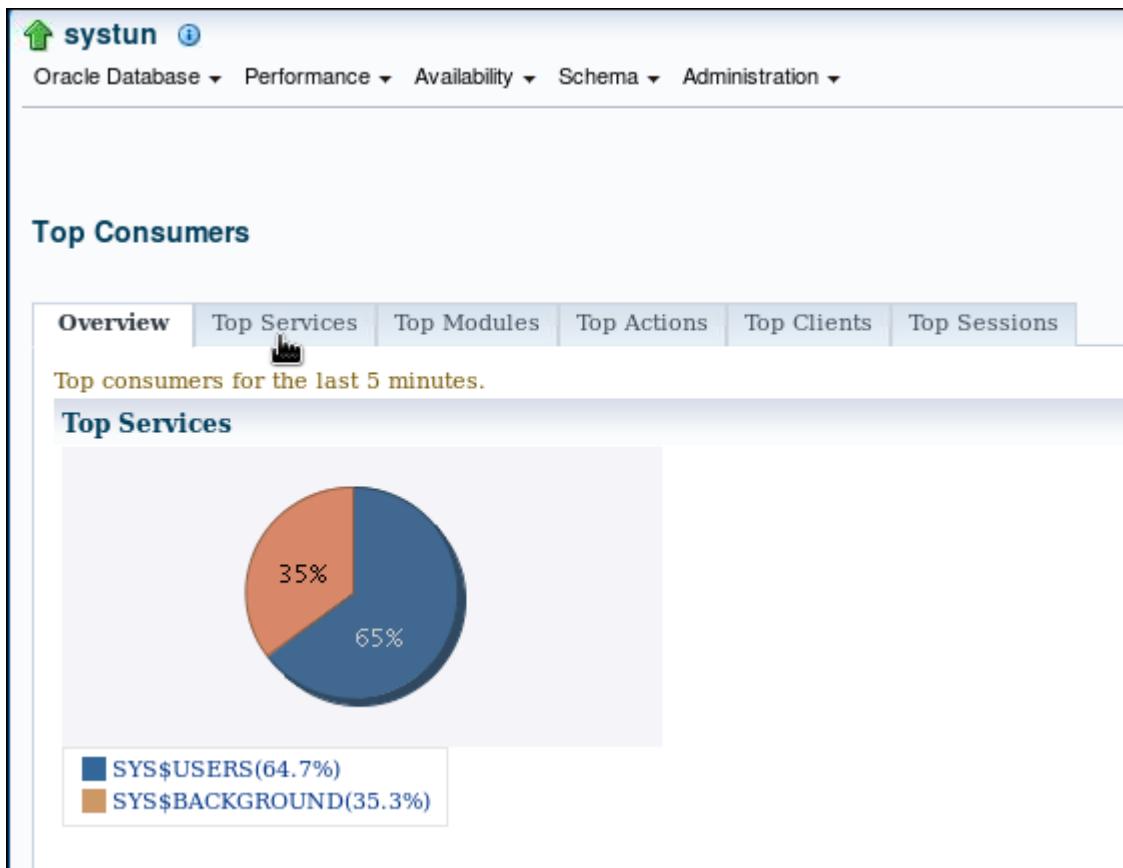
A screenshot of the 'Database Login' dialog box. It contains fields for 'Username' (DEV), 'Password' (redacted), 'Role' (NORMAL), and a 'Save As' checkbox. At the bottom are 'Login' and 'Cancel' buttons, with the cursor hovering over the 'Login' button.

- h. On the Performance page, click the **Top Consumers** link in the Additional Monitoring links section of the page.

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

- [Top Consumers](#)
- [Duplicate SQL](#)
- [Instance Locks](#)
- [Instance Activity](#)

- i. On the Top Consumers page, click the **Top Services** tab. This takes you to the Top Services page.



The screenshot shows the Oracle Enterprise Manager interface with the title 'systun'. The top navigation bar includes links for Enterprise, Targets, Favorites, and History. Below the title, there are tabs for Oracle Database, Performance, Availability, Schema, and Administration. A sub-menu for 'Top Consumers' is displayed.

The main content area is titled 'Top Consumers' and features a tab bar with 'Overview', 'Top Services' (which is selected), 'Top Modules', 'Top Actions', 'Top Clients', and 'Top Sessions'. A dropdown menu under 'View' is set to 'Active Services'. Below this are buttons for 'Enable SQL Trace', 'Disable SQL Trace', and 'View SQL Trace File'. There are also 'Select All' and 'Select None' checkboxes.

A table lists service activity:

| Select | Service | Activity (% for the last 5 minutes) | SQL Trace Enabled | Delta Elapsed Time (seconds) | Cumulative Ela |
|--------------------------|-----------------|-------------------------------------|-------------------|------------------------------|----------------|
| <input type="checkbox"/> | SYS\$USERS | 70.6 | FALSE | 0 | |
| <input type="checkbox"/> | SYS\$BACKGROUND | 29.4 | FALSE | 0 | |

Below the table, a section titled 'Additional Monitoring Links' contains a link to 'Top Activity page' and a bulleted list: 'Duplicate SQL' and 'Instance Locks'.

8. You now execute seven workload scripts that are traced. All workload scripts run under the TRACESERV service. Your goal is to analyze the generated trace files to interpret what happens in the seven cases. From your terminal session, execute the `run_tracep0.sh` script. This script is used to trigger the generation of statistics for your TRACESERV service so it can be viewed from within Enterprise Manager. As soon as you start the execution of the `run_tracep0.sh` script, move to the next step of this practice.

```
$ ./run_tracep0.sh
...
Connected to:...

SQL>
SQL> connect trace/trace@TRACESERV
Connected.
SQL>
SQL> alter session set tracefile_identifier='mytraceP0';

Session altered.

SQL>
SQL> set termout off
SQL>
SQL> exit;
```

- ...
- \$
9. Go back to the Top Consumers page in your Enterprise Manager session. Wait until you see the TRACESERV service in the Active Services table, and enable tracing for that service.
 - a. When you see TRACESERV in the Active Services table on the Top Services page, select it, and click the Enable SQL Trace button.

The screenshot shows the Oracle Enterprise Manager interface. The top navigation bar has tabs: Overview, Top Services (which is selected), Top Modules, Top Actions, Top Clients, and Top Sessions. Below the tabs, there is a dropdown menu labeled "View" with "Active Services" selected. Underneath the dropdown are three buttons: "Enable SQL Trace" (highlighted with a mouse cursor), "Disable SQL Trace", and "View SQL Trace File". A checkbox labeled "Select All" is checked, and "Select None" is available. The main area is a table titled "Active Services" with the following columns: Select, Service, Activity (% for the last 5 minutes), SQL Trace Enabled, and Delta Elapsed Time (seconds). The table contains three rows:

| Select | Service | Activity (% for the last 5 minutes) | SQL Trace Enabled | Delta Elapsed Time (seconds) |
|-------------------------------------|-----------------|-------------------------------------|-------------------|------------------------------|
| <input type="checkbox"/> | SYS\$BACKGROUND | 61.5 | FALSE | 0 |
| <input checked="" type="checkbox"/> | TRACESERV | 23.1 | FALSE | 0 |
| <input type="checkbox"/> | SYS\$USERS | 15.4 | FALSE | 0 |

- b. On the Enable SQL Trace page, make sure that Waits is set to TRUE, and Binds is set to FALSE. Click **OK**.

The screenshot shows the "Enable SQL Trace" dialog box. At the top, it says "Top Consumers > Enable SQL Trace" and "Logged in as DBSNMP". There are "Cancel" and "OK" buttons. The dialog title is "Enable SQL Trace". It has a table with three columns: Service, Waits, and Binds. The "Service" column has a dropdown menu with "TRACESERV" selected. The "Waits" column has a dropdown menu with "TRUE" selected. The "Binds" column has a dropdown menu with "FALSE" selected. At the bottom right of the dialog are "Cancel" and "OK" buttons, with the "OK" button highlighted by a mouse cursor.

- c. Back on the Top Services page, you should see a confirmation message near the top of the Top Consumers page.

The screenshot shows the Oracle Enterprise Manager Cloud Control 12c interface. At the top, there's a navigation bar with links for Enterprise, Targets, Favorites, and History. Below the navigation bar, the current user is listed as 'systun'. Underneath, there are tabs for Oracle Database, Performance, Availability, Schema, and Administration. A prominent yellow box with an information icon contains the message: 'Information: SQL Trace has been successfully enabled for the selected items.' At the bottom left, there's a section titled 'Top Consumers'.

- When tracing for TRACESERV is enabled and `run_tracep0.sh` has finished, execute the `run_tracep1.sh` script from your terminal session. Observe the Enterprise Manager Top Consumers/Top Services page.

```
$ ./run_tracep1.sh
...
SQL>
SQL> connect trace/trace@TRACESERV
Connected.
SQL>
SQL> alter session set workarea_size_policy=manual;

Session altered.

SQL>
SQL> alter session set sort_area_size=50000;

Session altered.

SQL>
SQL> alter session set hash_area_size=5000;

Session altered.

SQL>
SQL>
SQL> alter session set tracefile_identifier='mytraceP1';

Session altered.

SQL>
```

```
SQL>
SQL> set timing on
SQL>
SQL> select /*+ ORDERED USE_HASH(s2) */ count(*) from sales s1, sales
      s2 where s1.cust_id=s2.cust_id;

      COUNT(*)
-----
172878975

Elapsed: 00:01:19.25
SQL>
SQL>
SQL>
SQL> connect trace/trace@TRACESERV
Connected.
SQL>
SQL> alter session set tracefile_identifier='mytraceS1';

Session altered.

Elapsed: 00:00:00.00
SQL>
SQL> set timing on
SQL>
SQL> select /*+ ORDERED USE_HASH(s2) S1 */ count(*) from sales s1,
      sales s2 where s1.cust_id=s2.cust_id;

      COUNT(*)
-----
172878975

Elapsed: 00:00:40.19
SQL>
SQL> exit;

...
$
```

11. Execute the `run_tracep2.sh` script from your terminal session. Observe the output.

```
$ ./run_tracep2.sh
...
SQL*Plus: Release 12.1.0.1.0 Production on Tue Jul 24 22:23:59 2013
```

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

```
Copyright (c) 1982, 2013, Oracle. All rights reserved.

Connected to:
Oracle Database 11g Enterprise Edition Release 12.1.0.1.0- Production
With the Partitioning, OLAP, Data Mining and Real Application Testing
options

SQL>
SQL> connect trace/trace@TRACESERV
Connected.
SQL>
SQL> alter session set tracefile_identifier='mytraceP2';

Session altered.

SQL>
SQL> set timing on
SQL>
SQL> declare
  2      c      number := dbms_sql.open_cursor;
  3      oname  varchar2(50);
  4      ignore integer;
  5  begin
  6      for i in 1 .. 5000 loop
  7          dbms_sql.parse(c,'select object_name from dba_objects where
object_id = ''||i , dbms_sql.native); -- use literal
  8          dbms_sql.define_column(c, 1, oname, 50);
  9          ignore := dbms_sql.execute(c);
 10          if dbms_sql.fetch_rows(c)>0 then
 11              dbms_sql.column_value(c, 1, oname);
 12          end if;
 13      end loop;
 14      dbms_sql.close_cursor(c);
 15  end;
 16 /

PL/SQL procedure successfully completed.

Elapsed: 00:01:01.11
SQL>
SQL>
SQL>
SQL> connect trace/trace@TRACESERV
Connected.
```

```
SQL>
SQL> alter session set tracefile_identifier='mytraceS2';

Session altered.

Elapsed: 00:00:00.00
SQL>
SQL> declare
 2      c number := dbms_sql.open_cursor;
 3      oname  varchar2(50);
 4      ignore integer;
 5  begin
 6      dbms_sql.parse(c,'select object_name from dba_objects where
object_id = :y' , dbms_sql.native); -- use bind var
 7      for i in 1 .. 5000 loop
 8          dbms_sql.bind_variable(c,:y',i);
 9          dbms_sql.define_column(c, 1, oname, 50);
10          ignore := dbms_sql.execute(c);
11          if dbms_sql.fetch_rows(c)>0 then
12              dbms_sql.column_value(c, 1, oname);
13          end if;
14      end loop;
15      dbms_sql.close_cursor(c);
16  end;
17 /

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.40
SQL>
SQL> exit; Disconnected ...

$
```

12. Execute the `run_tracep3.sh` script from your terminal session. Observe the output.

```
$ ./run_tracep3.sh
...
SQL>
SQL> connect trace/trace@TRACESERV
Connected.
SQL>
SQL> alter session set tracefile_identifier='mytraceP3';
```

```
Session altered.

SQL>
SQL> update sales set amount_sold=20000 where prod_id=13 and
cust_id=987;

2 rows updated.

SQL>
SQL> commit;

Commit complete.

SQL>
SQL>
SQL> connect trace/trace
Connected.
SQL>
SQL> create index sales_prod_cust_indx on sales(prod_id,cust_id);

Index created.

SQL>
SQL> connect trace/trace@TRACESERV
Connected.
SQL>
SQL> alter session set tracefile_identifier='mytraceS3';

Session altered.

SQL>
SQL> update sales set amount_sold=30000 where prod_id=13 and
cust_id=987;

2 rows updated.

SQL>
SQL> commit;

Commit complete.

SQL>
SQL> connect trace/trace
Connected.
SQL>
SQL> drop index sales_prod_cust_indx;
```

```
Index dropped.
```

```
SQL>
SQL>
SQL> exit;
Disconnected ...
$
```

13. Execute the `run_tracep4.sh` script from your terminal session. Observe the output.

```
$ ./run_tracep4.sh
...
SQL>
SQL> connect trace/trace@TRACESERV
Connected.
SQL>
SQL> alter session set tracefile_identifier='mytraceP4';

Session altered.

SQL>
SQL> set timing on
SQL>
SQL> DECLARE
 2      TYPE SalesCurTyp  IS REF CURSOR;
 3      v_sales_cursor      SalesCurTyp;
 4      sales_record        sh.sales%ROWTYPE;
 5      v_stmt_str          VARCHAR2(200);
 6  BEGIN
 7      -- Dynamic SQL statement with placeholder:
 8      v_stmt_str := 'select * from sh.sales where amount_sold>0';
 9
10      -- Open cursor and specify bind argument in USING clause:
11      OPEN v_sales_cursor FOR v_stmt_str;
12
13      -- Fetch rows from result set one at a time:
14      LOOP
15          FETCH v_sales_cursor INTO sales_record;
16          EXIT WHEN v_sales_cursor%NOTFOUND;
17      END LOOP;
18
19      -- Close cursor:
20      CLOSE v_sales_cursor;
```

```
21  END;
22  /

PL/SQL procedure successfully completed.

Elapsed: 00:00:14.05
SQL>
SQL>
SQL> connect trace/trace@TRACESERV
Connected.
SQL>
SQL> alter session set tracefile_identifier='mytraceS4';

Session altered.

Elapsed: 00:00:00.00
SQL>
SQL> set timing on
SQL>
SQL> DECLARE
 2      TYPE SalesCurTyp  IS REF CURSOR;
 3      TYPE SalesList IS TABLE OF sh.sales%ROWTYPE;
 4      v_sales_cursor      SalesCurTyp;
 5      sales_List          SalesList;
 6      v_stmt_str          VARCHAR2(200);
 7  BEGIN
 8      -- Dynamic SQL statement with placeholder:
 9      v_stmt_str := 'select /* S4 */ * from sh.sales where
amount_sold>0';
10
11      -- Open cursor:
12      OPEN v_sales_cursor FOR v_stmt_str;
13
14      -- Fetch rows from result set one at a time:
15      LOOP
16          FETCH v_sales_cursor BULK COLLECT INTO Sales_List LIMIT
10000;
17          EXIT WHEN v_sales_cursor%NOTFOUND;
18      END LOOP;
19
20      -- Close cursor:
21      CLOSE v_sales_cursor;
22  END;
23  /
```

PL/SQL procedure successfully completed.

```
Elapsed: 00:00:01.80
SQL>
SQL>
SQL> exit;
Disconnected ...
$
```

14. Execute the `run_tracep5.sh` script from your terminal session. Observe the output.

```
$ ./run_tracep5.sh
...
Connected to: ...

SQL>
SQL> connect trace/trace@TRACESERV
Connected.
SQL>
SQL> alter session set tracefile_identifier='mytraceP5';

Session altered.

SQL>
SQL> insert into sales2 select * from sh.sales union all select * from
sales;

1837686 rows created.

SQL> commit;

Commit complete.

SQL>
SQL>
SQL> connect trace/trace@TRACESERV
Connected.
SQL>
SQL> alter session set tracefile_identifier='mytraceS5';

Session altered.

SQL>
SQL> insert into sales3 select * from sh.sales union all select * from
sales;
```

```
1837686 rows created.

SQL> commit;

Commit complete.

SQL>
SQL> exit;
Disconnected ...
$
```

15. Execute the `run_tracep6.sh` script from your terminal session. Observe the output. Wait until you see the message `run_tracep6 finished` before proceeding to the next step.

```
$ ./run_tracep6.sh
...
SQL>
SQL> connect trace/trace@TRACESERV
Connected.
SQL>
SQL> update sales set amount_sold=amount_sold+1;
Connected.
SQL>
SQL> alter session set tracefile_identifier='mytraceP6';

Session altered.

SQL>
SQL> exec dbms_lock.sleep(30);

918843 rows updated.

SQL>
SQL> exec dbms_lock.sleep(60);

SQL> exit;
Disconnected from Oracle Database 12c Enterprise Edition Release
12.1.0.1.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application
Testing options
[oracle@EDT3R13P1 Application_Tracing]$
PL/SQL procedure successfully completed.

SQL>
SQL> rollback;
```

```
Rollback complete.

SQL>
SQL> -- Run_tracep6 Finished
SQL> exit;
Disconnected ...

$
```

Note: The run_tracep6.sh takes awhile to execute completely. Wait for sometime to finish execution and then continue.

16. Execute the run_tracep7.sh script from your terminal session. Observe the output.

```
$ ./run_tracep7.sh
...
SQL>
SQL> connect trace/trace@TRACESERV
Connected.
SQL>
SQL> alter session set tracefile_identifier='mytraceP7';

Session altered.

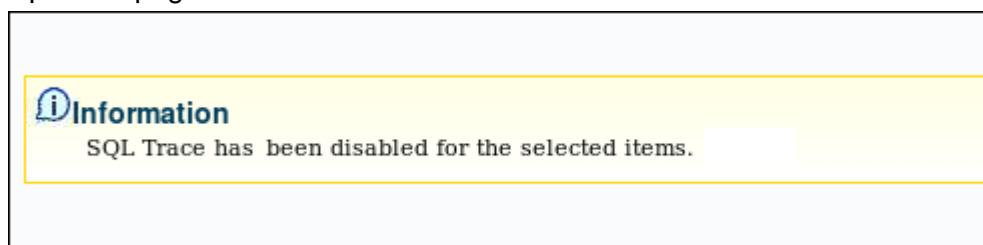
SQL>
SQL> declare
 2      c number := dbms_sql.open_cursor;
 3      custid number;
 4      amount number;
 5      ignore integer;
 6  begin
 7      dbms_sql.parse(c,'select cust_id, sum(amount_sold) from sales
where cust_id=2 group by cust_id order by cust_id' , dbms_sql.native);
-- use bind var
 8      dbms_sql.define_column(c, 1, custid);
 9      dbms_sql.define_column(c, 2, amount);
10      ignore := dbms_sql.execute(c);
11      if dbms_sql.fetch_rows(c)>0 then
12          dbms_sql.column_value(c, 1, custid);
13          dbms_sql.column_value(c, 2, amount);
14      end if;
15  end;
16  /
PL/SQL procedure successfully completed.

SQL>
SQL> connect trace/trace
```

```
Connected.  
SQL>  
SQL> create index sales_cust_indx on sales(cust_id);  
  
Index created.  
  
SQL>  
SQL> exit;  
Disconnected ...  
$
```

17. Disable tracing for your database.

- Go back to the Top Services page in your Enterprise Manager session.
- Select the **Services with Aggregation enable** from the drop down box of the View field.
- Ensure that TRACESERV is selected, and click the **Disable SQL trace** button.
- Back on the Top Consumers page, you should see a confirmation message near the top of the page.



18. Try to find out all the trace files that were generated to handle the previous seven cases.

Note: your output may differ based on your environment.

```
$ ./show_mytraces.sh  
systun_ora_19313_mytraceP1.trc  
systun_ora_19313_mytraceP1.trm  
systun_ora_19355_mytraceS1.trc  
systun_ora_19355_mytraceS1.trm  
systun_ora_19382_mytraceP2.trc  
systun_ora_19382_mytraceP2.trm  
systun_ora_19467_mytraceS2.trc  
systun_ora_19467_mytraceS2.trm  
systun_ora_19474_mytraceP3.trc  
systun_ora_19474_mytraceP3.trm  
systun_ora_19503_mytraceS3.trc  
systun_ora_19503_mytraceS3.trm  
systun_ora_19534_mytraceP4.trm  
systun_ora_19549_mytraceS4.trc  
systun_ora_19549_mytraceS4.trm
```

```
systun_ora_19558_mytraceP5.trc  
systun_ora_19558_mytraceP5.trm  
systun_ora_19568_mytraceS5.trc  
systun_ora_19568_mytraceS5.trm  
systun_ora_19583_mytraceP6.trc  
systun_ora_19583_mytraceP6.trm  
systun_ora_19634_mytraceP7.trc  
systun_ora_19634_mytraceP7.trm  
$
```

19. After you identify the location of those trace files, merge their content into one file called mytrace.trc located in your \$HOME/labs/solutions/Application_Tracing directory.

```
$ ./merge_traces.sh  
$
```

20. Use tkprof over the mytrace.trc file to generate a compiled trace output called myreport.txt located in your \$HOME/labs/solutions/Application_Tracing directory.

```
$ ./tkprof_traces.sh  
  
TKPROF: Release 12.1.0.1.0- Development on Wed Jul 25 02:00:00 2013  
  
Copyright (c) 1982, 2013, Oracle and/or its affiliates. All rights reserved.  
$
```

21. In addition, run TKPROF over the trace file that was generated for case 7 (step 16) with the EXPLAIN option set to your TRACE account.

```
$ tkprof /u01/app/oracle/diag/rdbms/systun/systun/trace/*mytraceP7.trc  
myreport2.txt explain=trace/trace  
  
TKPROF: Release 12.1.0.1.0- Development on Wed Jul 25 02:00:48 2013  
  
Copyright (c) 1982, 2013, Oracle and/or its affiliates. All rights reserved.  
$
```

22. After this is done, interpret the trace generated for case 1 (step 10). Case 1 appears near the beginning of the myreport.txt file.

Suggestion: Use a text editor or pager that has a search capability. gedit, vi, and less are available on the Oracle University classroom machines. gedit provides a GUI.

Hint: Search for the first occurrence of the string 'ORDERED USE_HASH(s2)' and examine the trace. Then find the second occurrence and compare the trace to the first occurrence. What do you observe, and what are your conclusions?

Case 1 illustrates the consequences of using manually sized SQL area parameters, such as HASH_AREA_SIZE. The first statement was executed by using a very small HASH_AREA_SIZE value. The immediate consequence was the number of temporary segments needed to execute the statement. Later, the same statement was executed, but this time by using the default SQL area parameters, which were sized automatically by the system to handle the needs. You can see a high disk value as well as a high number of waits for temporary segments for the first execution, compared to the second one. Compare the statistics direct path read temp and direct path write temp in the two statement traces. These indicate activity from SQL work areas to temporary segments.

Note: The actual statistics will vary from the example shown.

```
SQL ID: 5h4bydz30hwf7
Plan Hash: 2354142265
select /*+ ORDERED USE_HASH(s2) */ count(*)
from
  sales s1, sales s2 where s1.cust_id=s2.cust_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        2     105.91    118.33    806501     8874        0          1
-----  -----  -----  -----  -----  -----  -----  -----
total       4     105.92    118.34    806501     8874        0          1
Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 95

Rows      Row Source Operation
-----
1  SORT AGGREGATE (cr=8874 pr=806501 pw=2524 time=0 us)
172878975  HASH JOIN (cr=8874 pr=806501 pw=2524 time=244058880 us
cost=463845 size=1196022750 card=119602275)
918843    TABLE ACCESS FULL SALES (cr=4437 pr=4434 pw=0 time=770629
us cost=1230 size=4594215 card=918843)
918843    TABLE ACCESS FULL SALES (cr=4437 pr=4433 pw=0 time=775236
us cost=1230 size=4594215 card=918843)

Elapsed times include waiting on following events:
Event waited on            Times   Max. Wait  Total Waited
-----  -----  -----  -----
SQL*Net message to client           2        0.00        0.00
Disk file operations I/O           2        0.00        0.00
```

```

db file sequential read                      1      0.01      0.01
direct path read                           88      0.08     1.16
direct path write temp                     2524      0.02     0.77
direct path read temp                     797634      0.14     8.46
SQL*Net message from client                2      0.00      0.00
*****
...
SQL ID: 3a5334n3vuu8y
Plan Hash: 2354142265
select /*+ ORDERED USE_HASH(s2) S1 */ count(*)
from
  sales s1, sales s2 where s1.cust_id=s2.cust_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        2     61.66     67.70     8866     8874          0          1
-----
total       4     61.66     67.71     8866     8874          0          1

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 95

Rows      Row Source Operation
-----
1   SORT AGGREGATE (cr=8882 pr=2 pw=0 time=0 us)
172878975   HASH JOIN (cr=8882 pr=2 pw=0 time=188874752 us cost=4314
size=1196022750 card=119602275)
918843      TABLE ACCESS FULL SALES (cr=4441 pr=2 pw=0 time=791235 us
cost=1230 size=4594215 card=918843)
918843      TABLE ACCESS FULL SALES (cr=4441 pr=0 pw=0 time=938296 us
cost=1230 size=4594215 card=918843)

Elapsed times include waiting on following events:
Event waited on            Times      Max. Wait  Total Waited
-----  -----  -----  -----
SQL*Net message to client           2      0.00      0.00
SQL*Net message from client         2      0.00      0.00
*****

```

23. Interpret the trace generated for case 2 (step 11). In this trace, the first statement of interest has the string “use literal”.

Note: This trace is very long. What do you observe, and what are your conclusions?

- a. For this case, an almost identical statement was run 5000 times, but each time a different literal was used to execute the query. This caused the system to parse almost the same statement 5000 times, which is extremely inefficient, although the time precision is too low to give accurate information about the parse time of each statement. But the elapsed time and the CPU for the PL/SQL block include the time that these 5000 statements took to execute.

Note: Confirm that these 5000 executions are at “recursive depth: 1”.

The actual statistics you see will vary from the example shown. The first two recursive SQL statements are shown and then the last statement where object_id = 5000 is shown.

```
...
declare
    c      number := dbms_sql.open_cursor;
    oname  varchar2(50);
    ignore integer;
begin
    for i in 1 .. 5000 loop
        dbms_sql.parse(c,'select object_name from dba_objects where
object_id ='||i , dbms_sql.native); -- use literal
        dbms_sql.define_column(c, 1, oname, 50);
        ignore := dbms_sql.execute(c);
        if dbms_sql.fetch_rows(c)>0 then
            dbms_sql.column_value(c, 1, oname);
        end if;
    end loop;
    dbms_sql.close_cursor(c);
end;

call      count      cpu  elapsed       disk      query      current          rows
-----  -----  -----  -----  -----  -----  -----
Parse      1      0.00      0.00          0          0          0              0
Execute    1      4.27      5.77          0          0          0              1
Fetch      0      0.00      0.00          0          0          0              0
-----
total     2      4.27      5.77          0          0          0              1

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 95
```

```

Elapsed times include waiting on following events:
Event waited on                                Times   Max. Wait  Total Waited
-----      Waited
SQL*Net message to client                      1        0.00    0.00
SQL*Net message from client                    1        0.00    0.00
*****
```

SQL ID: 04qmn5w5qg1j3
 Plan Hash: 2729849777
 select object_name
 from
 dba_objects where object_id = 1

| call | count | cpu | elapsed | disk | query | current | rows |
|--------------|----------|-------------|-------------|----------|----------|----------|----------|
| Parse | 1 | 0.02 | 0.03 | 0 | 0 | 0 | 0 |
| Execute | 1 | 0.00 | 0.00 | 0 | 0 | 0 | 0 |
| Fetch | 1 | 0.00 | 0.00 | 1 | 3 | 0 | 0 |
| total | 3 | 0.02 | 0.03 | 1 | 3 | 0 | 0 |

Misses in library cache during parse: 1
 Optimizer mode: ALL_ROWS
 Parsing user id: 95 (**recursive depth: 1**)

| Rows | Row Source Operation |
|------|--|
| 0 | VIEW DBA_OBJECTS (cr=3 pr=1 pw=0 time=0 us cost=5 size=158 card=2) |
| 0 | UNION-ALL (cr=3 pr=1 pw=0 time=0 us) |
| 0 | FILTER (cr=3 pr=1 pw=0 time=0 us) |
| 0 | NESTED LOOPS (cr=3 pr=1 pw=0 time=0 us cost=5 size=108 card=1) |
| 0 | NESTED LOOPS (cr=3 pr=1 pw=0 time=0 us cost=4 size=104 card=1) |
| 0 | TABLE ACCESS BY INDEX ROWID OBJ\$ (cr=3 pr=1 pw=0 time=0 us cost=3 size=82 card=1) |
| 1 | INDEX RANGE SCAN I_OBJ1 (cr=2 pr=0 pw=0 time=0 us cost=2 size=0 card=1) (object id 36) |
| 0 | INDEX RANGE SCAN I_USER2 (cr=0 pr=0 pw=0 time=0 us cost=1 size=22 card=1) (object id 47) |
| 0 | INDEX RANGE SCAN I_USER2 (cr=0 pr=0 pw=0 time=0 us cost=1 size=4 card=1) (object id 47) |
| 0 | TABLE ACCESS BY INDEX ROWID IND\$ (cr=0 pr=0 pw=0 time=0 us cost=2 size=8 card=1) |
| 0 | INDEX UNIQUE SCAN I_IND1 (cr=0 pr=0 pw=0 time=0 us cost=1 size=0 card=1) (object id 41) |

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

```

          0      NESTED LOOPS  (cr=0 pr=0 pw=0 time=0 us cost=2 size=29
card=1)
          0          INDEX FULL SCAN I_USER2 (cr=0 pr=0 pw=0 time=0 us cost=1
size=20 card=1) (object id 47)
          0          INDEX RANGE SCAN I_OBJ4 (cr=0 pr=0 pw=0 time=0 us cost=1
size=9 card=1) (object id 39)
          0          FILTER   (cr=0 pr=0 pw=0 time=0 us)
          0      NESTED LOOPS  (cr=0 pr=0 pw=0 time=0 us cost=1 size=83
card=1)
          0          INDEX FULL SCAN I_LINK1 (cr=0 pr=0 pw=0 time=0 us cost=0
size=79 card=1) (object id 137)
          0          INDEX RANGE SCAN I_USER2 (cr=0 pr=0 pw=0 time=0 us cost=1
size=4 card=1) (object id 47)

Elapsed times include waiting on following events:
Event waited                      Times     Max. Wait  Total Waited
-----  Waited
db file sequential read           1          0.00        0.00
*****
SQL ID: 28m8r9uth07db
Plan Hash: 2729849777
select object_name
from
  dba_objects where object_id = 2

call    count      cpu  elapsed       disk    query    current      rows
-----  -----  -----
Parse      1      0.01      0.02          0        0          0          0
Execute    1      0.00      0.00          0        0          0          0
Fetch      1      0.00      0.00          0        5          0          1
-----  -----  -----
total      3      0.01      0.02          0        5          0          1

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 95      (recursive depth: 1)

Rows      Row Source Operation
-----
1  VIEW   DBA_OBJECTS (cr=5 pr=0 pw=0 time=0 us cost=5 size=158
card=2)
1  UNION-ALL (cr=5 pr=0 pw=0 time=0 us)
1  FILTER  (cr=5 pr=0 pw=0 time=0 us)

```

```

      1      NESTED LOOPS  (cr=5 pr=0 pw=0 time=0 us cost=5 size=108
card=1)
      1      NESTED LOOPS  (cr=4 pr=0 pw=0 time=0 us cost=4 size=104
card=1)
      1      TABLE ACCESS BY INDEX ROWID OBJ$ (cr=3 pr=0 pw=0 time=0
us cost=3 size=82 card=1)
      1          INDEX RANGE SCAN I_OBJ1 (cr=2 pr=0 pw=0 time=0 us
cost=2 size=0 card=1) (object id 36)
      1          INDEX RANGE SCAN I_USER2 (cr=1 pr=0 pw=0 time=0 us
cost=1 size=22 card=1) (object id 47)
      1          INDEX RANGE SCAN I_USER2 (cr=1 pr=0 pw=0 time=0 us cost=1
size=4 card=1) (object id 47)
      0      TABLE ACCESS BY INDEX ROWID IND$ (cr=0 pr=0 pw=0 time=0 us
cost=2 size=8 card=1)
      0          INDEX UNIQUE SCAN I_IND1 (cr=0 pr=0 pw=0 time=0 us cost=1
size=0 card=1) (object id 41)
      0      NESTED LOOPS  (cr=0 pr=0 pw=0 time=0 us cost=2 size=29
card=1)
      0          INDEX FULL SCAN I_USER2 (cr=0 pr=0 pw=0 time=0 us cost=1
size=20 card=1) (object id 47)
      0          INDEX RANGE SCAN I_OBJ4 (cr=0 pr=0 pw=0 time=0 us cost=1
size=9 card=1) (object id 39)
      0          FILTER   (cr=0 pr=0 pw=0 time=0 us)
      0      NESTED LOOPS  (cr=0 pr=0 pw=0 time=0 us cost=1 size=83
card=1)
      0          INDEX FULL SCAN I_LINK1 (cr=0 pr=0 pw=0 time=0 us cost=0
size=79 card=1) (object id 137)
      0          INDEX RANGE SCAN I_USER2 (cr=0 pr=0 pw=0 time=0 us cost=1
size=4 card=1) (object id 47)

*****
...
```

SQL ID: 2d2078j2pucd5
 Plan Hash: 2729849777
 select object_name
 from
 dba_objects where object_id = 5000

| call | count | cpu | elapsed | disk | query | current | rows |
|--------------|----------|-------------|-------------|----------|----------|----------|----------|
| Parse | 1 | 0.01 | 0.01 | 0 | 0 | 0 | 0 |
| Execute | 1 | 0.00 | 0.00 | 0 | 0 | 0 | 0 |
| Fetch | 1 | 0.00 | 0.00 | 0 | 5 | 0 | 1 |
| total | 3 | 0.01 | 0.01 | 0 | 5 | 0 | 1 |

```

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 95      (recursive depth: 1)

Rows          Row Source Operation
-----
   1  VIEW    DBA_OBJECTS (cr=5 pr=0 pw=0 time=0 us cost=5 size=158
card=2)
      1  UNION-ALL  (cr=5 pr=0 pw=0 time=0 us)
      1  FILTER   (cr=5 pr=0 pw=0 time=0 us)
      1  NESTED LOOPS (cr=5 pr=0 pw=0 time=0 us cost=5 size=108
card=1)
      1  NESTED LOOPS (cr=4 pr=0 pw=0 time=0 us cost=4 size=104
card=1)
         1  TABLE ACCESS BY INDEX ROWID OBJ$ (cr=3 pr=0 pw=0 time=0
us cost=3 size=82 card=1)
         1  INDEX RANGE SCAN I_OBJ1 (cr=2 pr=0 pw=0 time=0 us
cost=2 size=0 card=1) (object id 36)
         1  INDEX RANGE SCAN I_USER2 (cr=1 pr=0 pw=0 time=0 us
cost=1 size=22 card=1) (object id 47)
         1  INDEX RANGE SCAN I_USER2 (cr=1 pr=0 pw=0 time=0 us cost=1
size=4 card=1) (object id 47)
         0  TABLE ACCESS BY INDEX ROWID IND$ (cr=0 pr=0 pw=0 time=0 us
cost=2 size=8 card=1)
         0  INDEX UNIQUE SCAN I_IND1 (cr=0 pr=0 pw=0 time=0 us cost=1
size=0 card=1) (object id 41)
         0  NESTED LOOPS (cr=0 pr=0 pw=0 time=0 us cost=2 size=29
card=1)
         0  INDEX FULL SCAN I_USER2 (cr=0 pr=0 pw=0 time=0 us cost=1
size=20 card=1) (object id 47)
         0  INDEX RANGE SCAN I_OBJ4 (cr=0 pr=0 pw=0 time=0 us cost=1
size=9 card=1) (object id 39)
         0  FILTER   (cr=0 pr=0 pw=0 time=0 us)
         0  NESTED LOOPS (cr=0 pr=0 pw=0 time=0 us cost=1 size=83
card=1)
         0  INDEX FULL SCAN I_LINK1 (cr=0 pr=0 pw=0 time=0 us cost=0
size=79 card=1) (object id 137)
         0  INDEX RANGE SCAN I_USER2 (cr=0 pr=0 pw=0 time=0 us cost=1
size=4 card=1) (object id 47)

*****

```

- b. Another statement was also executed 5000 times but, this time, using a bind variable. The statement of interest has the string “use bind var”. The statement was parsed only once, and executed 5000 times. This behavior is much more efficient than the previous one.

Note: The CPU and elapsed columns for the PL/SQL block include the time spent on all the SQL statements executed inside the block.

```

...
*****
declare
    c number := dbms_sql.open_cursor;
    oname varchar2(50);
    ignore integer;
begin
    dbms_sql.parse(c,'select object_name from dba_objects where
object_id = :y' , dbms_sql.native); -- use bind var
    for i in 1 .. 5000 loop
        dbms_sql.bind_variable(c,:y',i);
        dbms_sql.define_column(c, 1, oname, 50);
        ignore := dbms_sql.execute(c);
        if dbms_sql.fetch_rows(c)>0 then
            dbms_sql.column_value(c, 1, oname);
        end if;
    end loop;
    dbms_sql.close_cursor(c);
end;

call      count      cpu  elapsed       disk    query     current         rows
-----  -----  -----  -----  -----  -----  -----
Parse        1      0.01      0.01        0          0          0          0
Execute      1      0.55      0.55        0          0          0          1
Fetch        0      0.00      0.00        0          0          0          0
-----  -----  -----  -----  -----  -----
total       2      0.56      0.56        0          0          0          1

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 95

Elapsed times include waiting on following events:
Event waited on                                Times     Max. Wait  Total Waited
-----  -----  -----  -----
SQL*Net message to client                      1          0.00    0.00
SQL*Net message from client                    1          0.00    0.00
*****
SQL ID: 25t3qdy59b8tk
Plan Hash: 2729849777
select object_name
from
    dba_objects where object_id = :y

```

| call | count | cpu | elapsed | disk | query | current | rows |
|--------------|--------------|-------------|-------------|----------|--------------|----------|-------------|
| Parse | 1 | 0.00 | 0.00 | 0 | 0 | 0 | 0 |
| Execute | 5000 | 0.23 | 0.22 | 0 | 0 | 0 | 0 |
| Fetch | 5000 | 0.27 | 0.29 | 0 | 26810 | 0 | 4931 |
| total | 10001 | 0.50 | 0.51 | 0 | 26810 | 0 | 4931 |

Misses in library cache during parse: 1
 Misses in library cache during execute: 1
 Optimizer mode: ALL_ROWS
 Parsing user id: 95 (**recursive depth: 1**)

| Rows | Row Source Operation |
|------|--|
| 0 | VIEW DBA_OBJECTS (cr=3 pr=0 pw=0 time=0 us cost=5 size=158 card=2) |
| 0 | UNION-ALL (cr=3 pr=0 pw=0 time=0 us) |
| 0 | FILTER (cr=3 pr=0 pw=0 time=0 us) |
| 0 | NESTED LOOPS (cr=3 pr=0 pw=0 time=0 us cost=5 size=108 card=1) |
| 0 | NESTED LOOPS (cr=3 pr=0 pw=0 time=0 us cost=4 size=104 card=1) |
| 0 | TABLE ACCESS BY INDEX ROWID OBJ\$ (cr=3 pr=0 pw=0 time=0 us cost=3 size=82 card=1) |
| 1 | INDEX RANGE SCAN I_OBJ1 (cr=2 pr=0 pw=0 time=0 us cost=2 size=0 card=1) (object id 36) |
| 0 | INDEX RANGE SCAN I_USER2 (cr=0 pr=0 pw=0 time=0 us cost=1 size=22 card=1) (object id 47) |
| 0 | INDEX RANGE SCAN I_USER2 (cr=0 pr=0 pw=0 time=0 us cost=1 size=4 card=1) (object id 47) |
| 0 | TABLE ACCESS BY INDEX ROWID IND\$ (cr=0 pr=0 pw=0 time=0 us cost=2 size=8 card=1) |
| 0 | INDEX UNIQUE SCAN I_IND1 (cr=0 pr=0 pw=0 time=0 us cost=1 size=0 card=1) (object id 41) |
| 0 | NESTED LOOPS (cr=0 pr=0 pw=0 time=0 us cost=2 size=29 card=1) |
| 0 | INDEX FULL SCAN I_USER2 (cr=0 pr=0 pw=0 time=0 us cost=1 size=20 card=1) (object id 47) |
| 0 | INDEX RANGE SCAN I_OBJ4 (cr=0 pr=0 pw=0 time=0 us cost=1 size=9 card=1) (object id 39) |
| 0 | FILTER (cr=0 pr=0 pw=0 time=0 us) |
| 0 | NESTED LOOPS (cr=0 pr=0 pw=0 time=0 us cost=1 size=83 card=1) |
| 0 | INDEX FULL SCAN I_LINK1 (cr=0 pr=0 pw=0 time=0 us cost=0 size=79 card=1) (object id 137) |
| 0 | INDEX RANGE SCAN I_USER2 (cr=0 pr=0 pw=0 time=0 us cost=1 size=4 card=1) (object id 47) |

24. Interpret the trace generated for case 3 (step 12). This trace starts at the string “update sales set amount_sold=20000”. What do you observe, and what are your conclusions?

If you look closely at this case, you see that you access the complete table to update only two rows. This is very inefficient. The alternative case is much better because it uses an index to speed up the retrieval of the rows that need to be updated.

```
SQL ID: 8mt8gqs6btkb6
Plan Hash: 3654535892
update sales set amount_sold=20000
where
  prod_id=13 and cust_id=987

call      count        cpu   elapsed         disk    query     current          rows
-----  -----  -----  -----  -----  -----  -----  -----
Parse       1        0.00      0.00          0        1        0        0
Execute     1        0.09      0.10      228    4441        3        2
Fetch       0        0.00      0.00          0        0        0        0
-----  -----  -----  -----  -----  -----  -----  -----
total      2        0.10      0.11      228    4442        3        2

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 95

Rows      Row Source Operation
-----
 0  UPDATE  SALES (cr=4441 pr=228 pw=0 time=0 us)
 2  TABLE ACCESS FULL SALES (cr=4441 pr=228 pw=0 time=19 us
cost=1231 size=3549 card=91)

Elapsed times include waiting on following events:
Event waited on                      Times     Max. Wait  Total Waited
-----  -----  -----  -----
db file scattered read                20        0.00        0.00
db file sequential read              184        0.00        0.00
SQL*Net message to client            1        0.00        0.00
SQL*Net message from client          1        0.00        0.00
...
SQL ID: c9ffzgn5fv6gk
```

```

Plan Hash: 1889391443
update sales set amount_sold=30000
where
prod_id=13 and cust_id=987

call      count        cpu  elapsed       disk    query     current         rows
-----  -----  -----  -----  -----  -----  -----  -----
Parse        1        0.00  0.00          0        2          0          0
Execute      1        0.00  0.00          0        3          3          2
Fetch        0        0.00  0.00          0        0          0          0
-----  -----  -----  -----  -----  -----  -----  -----
total       2        0.00  0.00          0        5          3          2

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 95

Rows      Row Source Operation
-----
0  UPDATE  SALES (cr=3 pr=0 pw=0 time=0 us)
2  INDEX RANGE SCAN SALES_PROD_CUST_INDX (cr=3 pr=0 pw=0 time=2
us cost=3 size=78 card=2) (object id 78771)

Elapsed times include waiting on following events:
Event waited on                      Times     Max. Wait  Total Waited
-----  -----  -----  -----
SQL*Net message to client            1          0.00      0.00
SQL*Net message from client          1          0.01      0.01
*****
```

25. Interpret the trace generated for case 4 (step 13). This trace can be found below the string mytraceP4. What do you observe, and what are your conclusions?

In this case, the first statement does not use the fetching mechanism appropriately. One fetch operation is done for every single row retrieved. This is also very inefficient. The alternative case is doing 92 fetches to retrieve the same amount of rows. This technique is called array fetch.

```

SQL ID: 1j17cdqu55s6k
Plan Hash: 0
alter session set tracefile_identifier='mytraceP4'
...
SQL ID: fh0354r530g32
Plan Hash: 1550251865
select *
from
```

```
sh.sales where amount_sold>0

call      count      cpu    elapsed      disk      query      current      rows
-----  -----
Parse        1      0.05      0.05          0          0          0          0
Execute      1      0.00      0.00          0          0          0          0
Fetch  918844     8.98     10.80       879  918942          0      918843
-----  -----
total   918846     9.04     10.86       879  918942          0      918843

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 95      (recursive depth: 1)

Rows      Row Source Operation
-----
918843 PARTITION RANGE ALL PARTITION: 1 28 (cr=918942 pr=879 pw=0
time=7298636 us cost=490 size=26646447 card=918843)
918843 TABLE ACCESS FULL SALES PARTITION: 1 28 (cr=918942 pr=879
pw=0 time=4239558 us cost=490 size=26646447 card=918843)

Elapsed times include waiting on following events:
Event waited on                      Times      Max. Wait  Total Waited
-----  -----
Waited
Disk file operations I/O                  1          0.00      0.00
db file sequential read                 52          0.05      0.32
db file scattered read                122          0.07      1.33
*****
...
```

SQL ID: ftygqy235kbwv
Plan Hash: 1550251865
select /* S4 */ *
from
sh.sales where amount_sold>0

| 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 | 92 | 2.47 | 2.52 | 0 | 1809 | 0 | 918843 |

```

-----+
total      94     2.48     2.52      0    1809      0   918843

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 95      (recursive depth: 1)

Rows          Row Source Operation
-----
918843  PARTITION RANGE ALL PARTITION: 1 28 (cr=1809 pr=0 pw=0
time=2810666 us cost=490 size=26646447 card=918843)
918843  TABLE ACCESS FULL SALES PARTITION: 1 28 (cr=1809 pr=0 pw=0
time=1190617 us cost=490 size=26646447 card=918843)

Elapsed times include waiting on following events:
Event waited on                      Times     Max. Wait  Total Waited
-----                               Waited
Disk file operations I/O                  1        0.00      0.00
*****
```

26. Interpret the trace generated for case 5 (step 14). This trace can be found using the string `union all`. What do you observe, and what are your conclusions?

In this statement, notice the large values associated with the Execute phase. Here, the first statement incurs too many recursive calls to allocate extents to the table. This is because the tablespace in which it is stored is not correctly set up for extent allocations. The symptoms can be seen in the number of statements that follow this statement that have the message “recursive depth: 1.” These recursive statements are not seen if `SYS=NO` in the `tkprof` command.

The alternative case, which starts with the next statement containing the string `union all`, is much better, as you can see. The times for the execute phase are much lower compared to the first case. Also, the number of recursive statements in the second case should be much lower than the first one.

```

SQL ID: 4u687pw6m3kst
Plan Hash: 3659198364
insert into sales2 select * from sh.sales union all select * from
sales

call      count      cpu    elapsed      disk      query      current      rows
-----  -----  -----  -----  -----  -----  -----  -----
Parse        1      0.01      0.03          1          3          0          0
Execute      1      3.04      4.09         13      9871     66966      782730
Fetch        0      0.00      0.00          0          0          0          0
-----  -----  -----  -----  -----  -----  -----  -----
total       2      3.06      4.13         14      9874     66966      782730
```

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

```

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 95

Rows          Row Source Operation
-----
0   LOAD TABLE CONVENTIONAL (cr=0 pr=0 pw=0 time=0 us)
782731    UNION-ALL (cr=1448 pr=0 pw=0 time=5177587 us)
782731      PARTITION RANGE ALL PARTITION: 1 28 (cr=1448 pr=0 pw=0
time=2384664 us cost=489 size=26646447 card=918843)
782731        TABLE ACCESS FULL SALES PARTITION: 1 28 (cr=1448 pr=0 pw=0
time=1087553 us cost=489 size=26646447 card=918843)
0     TABLE ACCESS FULL SALES (cr=0 pr=0 pw=0 time=0 us cost=1233
size=78831570 card=906110)

Elapsed times include waiting on following events:
Event waited on                                Times     Max. Wait  Total Waited
-----  Waited
Disk file operations I/O                         1          0.00    0.00
db file sequential read                        13          0.02    0.09
log file switch completion                     2          0.04    0.05
log file sync                               1          0.00    0.00
SQL*Net break/reset to client                 2          0.01    0.01
SQL*Net message to client                      1          0.00    0.00
SQL*Net message from client                   1          0.22    0.22
*****
SQL ID: bsa0wjtftg3uw
Plan Hash: 1512486435
select file#
from
file$ where ts#:=1

call      count      cpu  elapsed       disk      query  current          rows
-----  -----  -----  -----  -----  -----  -----  -----
Parse      769      0.02      0.02          0          0          0          0
Execute     769      0.02      0.02          0          0          0          0
Fetch     1538      0.02      0.03          0      3072          0      769
-----  -----  -----  -----  -----  -----  -----  -----
total     3076      0.07      0.08          0      3072          0      769

Misses in library cache during parse: 1
Misses in library cache during execute: 1
Optimizer mode: CHOOSE

```

```

Parsing user id: SYS      (recursive depth: 1)

Rows      Row Source Operation
-----
   1  TABLE ACCESS FULL FILE$ (cr=4 pr=0 pw=0 time=0 us cost=2
size=6 card=1)

*****
SQL ID: 0kkhhb2w93cx0
Plan Hash: 2170058777
update seg$ set
type#:=4,blocks=:5,extents=:6,minexts=:7,maxexts=:8,extsize=
:9,extpct=:10,user#:=11,in/exts=:12,lists=decode(:13, 65535, NULL,
:13),
groups=decode(:14, 65535, NULL, :14), cachehint=:15, hwmincr=:16,
spare1=
DECODE(:17,0,NULL,:17), scanhint=:18, bitmappranges=:19
where
ts#:=1 and file#:=2 and block#:=3

call      count      cpu    elapsed      disk      query      current      rows
-----  -----  -----  -----  -----  -----  -----  -----
Parse      763      0.02      0.01          0          0          0          0
Execute     763      0.16      0.16          0      2289        801        763
Fetch         0      0.00      0.00          0          0          0          0
-----  -----  -----  -----  -----  -----  -----  -----
total     1526      0.19      0.18          0      2289        801        763

Misses in library cache during parse: 1
Misses in library cache during execute: 1
Optimizer mode: CHOOSE
Parsing user id: SYS      (recursive depth: 1)

Rows      Row Source Operation
-----
   0  UPDATE  SEG$ (cr=3 pr=0 pw=0 time=0 us)
   1  TABLE ACCESS CLUSTER SEG$ (cr=3 pr=0 pw=0 time=0 us cost=2
size=68 card=1)
   1  INDEX UNIQUE SCAN I_FILE#_BLOCK# (cr=2 pr=0 pw=0 time=0 us
cost=1 size=0 card=1) (object id 9)
*****
...
SQL ID: cxnuy9bystsx5
Plan Hash: 0

```

```

alter session set tracefile_identifier='mytraceS5'
...
SQL ID: 0949qykm7qlt0
Plan Hash: 3659198364
insert into sales3 select * from sh.sales union all select * from
sales

call      count      cpu    elapsed      disk      query      current      rows
-----  -----  -----  -----  -----  -----  -----  -----
Parse        1      0.01      0.03          1          2          0          0
Execute      1      1.53      1.76         15      5534     22362      533382
Fetch        0      0.00      0.00          0          0          0          0
-----  -----  -----  -----  -----  -----  -----  -----
total       2      1.54      1.80         16      5536     22362      533382

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 95

Rows      Row Source Operation
-----
      0  LOAD TABLE CONVENTIONAL  (cr=0 pr=0 pw=0 time=0 us)
  533383  UNION-ALL  (cr=992 pr=0 pw=0 time=3671608 us)
  533383    PARTITION RANGE ALL PARTITION: 1 28 (cr=992 pr=0 pw=0
time=1746800 us cost=489 size=26646447 card=918843)
  533383    TABLE ACCESS FULL SALES PARTITION: 1 28 (cr=992 pr=0 pw=0
time=816851 us cost=489 size=26646447 card=918843)
      0    TABLE ACCESS FULL SALES (cr=0 pr=0 pw=0 time=0 us cost=1233
size=78831570 card=906110)

Elapsed times include waiting on following events:
Event waited on                      Times   Max. Wait  Total Waited
-----  -----  -----  -----
Disk file operations I/O                  2      0.00      0.00
db file sequential read                 15      0.01      0.06
log file sync                           1      0.00      0.00
SQL*Net break/reset to client           2      0.00      0.00
SQL*Net message to client                1      0.00      0.00
SQL*Net message from client              1      0.00      0.00
*****
```

27. Interpret the trace generated for case 6 (step 15). This trace starts at the string bxraux4u04and, which is the SQL ID for the statement. What do you observe and what are your conclusions?

This case is more difficult to understand. Here, you select a table that is entirely locked by another transaction. This forces the query to generate consistent read blocks for almost the entire table, causing undo segments to be accessed. This is shown in the statistics by the large query value, and almost no current blocks.

```
...
SQL ID: bxraux4u04and
Plan Hash: 3229864837
select cust_id, sum(amount_sold)
from
sales group by cust_id order by cust_id

call      count      cpu    elapsed      disk      query      current          rows
-----  -----  -----  -----  -----  -----  -----  -----
Parse        1      0.00      0.00          0          1          0          0
Execute       1      0.00      0.00          0          0          0          0
Fetch      472     3.17     3.45     4047  978282          0      7059
-----  -----  -----  -----  -----  -----  -----  -----
total      474     3.17     3.45     4047  978283          0      7059

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 95

Rows      Row Source Operation
-----
 7059  SORT GROUP BY (cr=978282 pr=4047 pw=0 time=9496 us cost=1259
size=23558860 card=906110)
 918843  TABLE ACCESS FULL SALES (cr=978282 pr=4047 pw=0 time=4061770
us cost=1233 size=23558860 card=906110)

Elapsed times include waiting on following events:
Event waited on                      Times      Max. Wait  Total Waited
-----  -----  -----  -----
SQL*Net message to client            472          0.00      0.00
Disk file operations I/O              1          0.00      0.00
db file sequential read            3679          0.00      0.03
db file scattered read                5          0.00      0.00
SQL*Net message from client         472          0.00      0.03
*****
*****
```

28. Interpret the trace generated for case 7 (step 16). What do you observe and what are your conclusions?

- a. For case 7, you should compare the content in `myreport.txt` with the content in `myreport2.txt`. You should see that an index was added after the first trace was generated. This situation can cause confusion, especially if the trace does not contain an execution plan to begin with. This is what you can see from within `myreport.txt`:

```
SQL ID: 51xxq509gnbbc
Plan Hash: 1729043503
select cust_id, sum(amount_sold)
from
  sales where cust_id=2 group by cust_id order by cust_id

call      count      cpu    elapsed      disk      query      current          rows
-----  -----  -----  -----  -----  -----  -----  -----
Parse        1      0.00      0.00      0      1      0      0
Execute      1      0.00      0.00      0      0      0      0
Fetch        1     1.75     1.77      1  978282      0      1
-----  -----  -----  -----  -----  -----  -----  -----
total       3     1.75     1.77      1  978283      0      1

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 95      (recursive depth: 1)

Rows      Row Source Operation
-----
1 RESULT CACHE gxd1gfy5dxhzzagytkv7nng9a2 (cr=978282 pr=1 pw=0
time=0 us)
1 SORT GROUP BY NOSORT (cr=978282 pr=1 pw=0 time=0 us
cost=1231 size=3718 card=143)
176 TABLE ACCESS FULL SALES (cr=978282 pr=1 pw=0 time=375637 us
cost=1231 size=3718 card=143)

Elapsed times include waiting on following events:
Event waited on          Times   Max. Wait  Total Waited
-----  -----  -----  -----
Disk file operations I/O           1      0.00      0.00
db file sequential read           1      0.00      0.00
*****
*****
```

- b. This is what you see from `myreport2.txt`. Notice that the row source section shows a TABLE ACCESS FULL and the explain plan shows a TABLE ACCESS BY INDEX ROWID. This is an indication that the explain plan is not accurate.

```

SQL ID: 51xxq509gnbbc
Plan Hash: 1729043503
select cust_id, sum(amount_sold)
from
  sales where cust_id=2 group by cust_id order by cust_id

call      count      cpu   elapsed      disk      query      current          rows
-----  -----  -----  -----  -----  -----  -----  -----
Parse        1      0.00      0.00          0          1          0          0
Execute       2      0.00      0.00          0          0          0          0
Fetch        1      1.75      1.77          1  978282          0          1
-----  -----  -----  -----  -----  -----  -----  -----
total        4      1.75      1.77          1  978283          0          1

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 95 (TRACE) (recursive depth: 1)

Rows      Row Source Operation
-----
         1 RESULT CACHE gxd1gfy5dxhzzagytkv7nng9a2 (cr=978282 pr=1 pw=0
time=0 us)
         1 SORT GROUP BY NOSORT (cr=978282 pr=1 pw=0 time=0 us
cost=1231 size=3718 card=143)
         176 TABLE ACCESS FULL SALES (cr=978282 pr=1 pw=0 time=375637 us
cost=1231 size=3718 card=143)

Rows      Execution Plan
-----
         0 SELECT STATEMENT MODE: ALL_ROWS
         1 RESULT CACHE OF 'gxd1gfy5dxhzzagytkv7nng9a2'
           column-count=2; type=AUTO; dependencies=(TRACE.SALES);
name=
           "select cust_id, sum(amount_sold) from sales where
cust_id=2
           group by cust_id order by cust_id"
         1 SORT (GROUP BY NOSORT)
         176 TABLE ACCESS MODE: ANALYZED (BY INDEX ROWID) OF 'SALES'
           (TABLE)
         0 INDEX MODE: ANALYZED (RANGE SCAN) OF 'SALES_CUST_INDX'
           (INDEX)

Elapsed times include waiting on following events:
Event waited on          Times     Max. Wait    Total Waited

```

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

| | Waited | | |
|-----------------------------|--------|------|------|
| SQL*Net message to client | 1 | 0.00 | 0.00 |
| SQL*Net message from client | 1 | 0.00 | 0.00 |
| Disk file operations I/O | 1 | 0.00 | 0.00 |
| db file sequential read | 1 | 0.00 | 0.00 |
| ***** | | | |

29. You can now clean up your environment by executing the `at_cleanup.sh` script from your terminal window.

```
$ ./at_cleanup.sh

SQL> exec DBMS_SERVICE.STOP_SERVICE('TRACESERV');

PL/SQL procedure successfully completed.

SQL>
SQL> exec DBMS_SERVICE.DELETE_SERVICE('TRACESERV');

PL/SQL procedure successfully completed.

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

Practices for Lesson 4: Using Basic Techniques

Chapter 4

Practices for Lesson 4: Overview

Practices Overview

In this practice, you perform the following:

- Rewriting queries for better performance
- Rewriting applications for better performance

List of Cases:

- Case 1: Correlated Subquery
- Case 2: Join Conditions
- Case 3: Simple Predicate
- Case 4: Union
- Case 5: Combining SQL Statements
- Case 6: Database Connection Management

Practice 4: Avoiding Common Mistakes

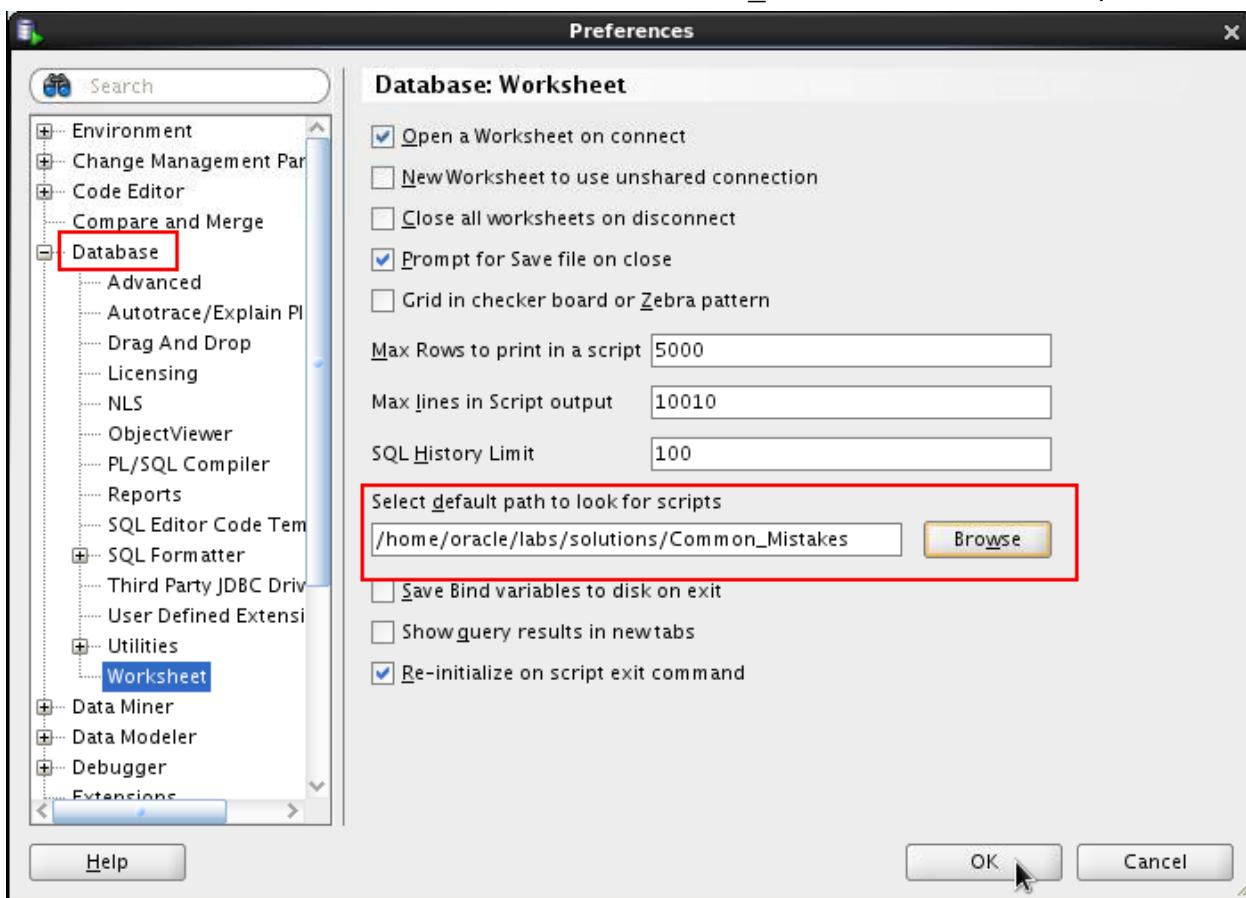
Overview

In this practice, you examine some common mistakes in writing SQL statements. You have to find a workaround to enhance performance.

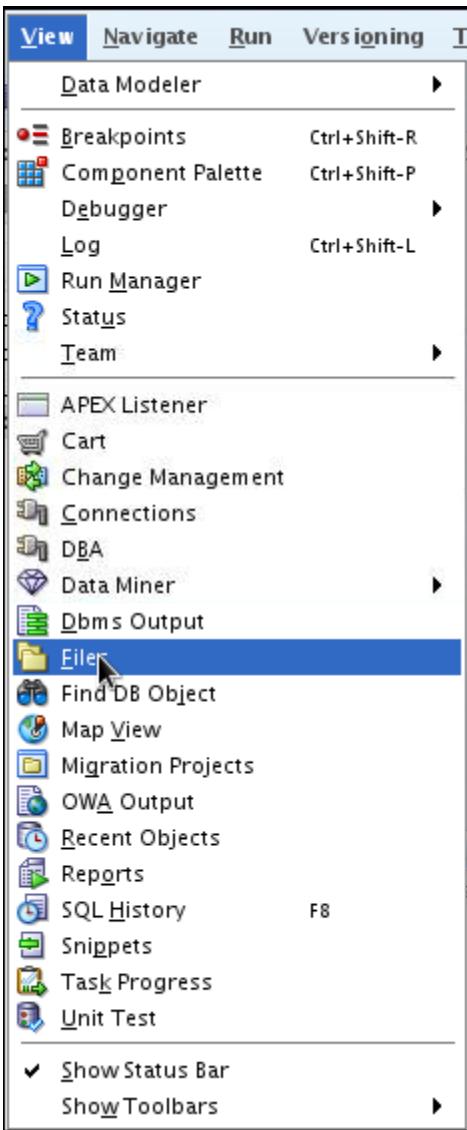
Tasks

Case 1: Correlated Subquery

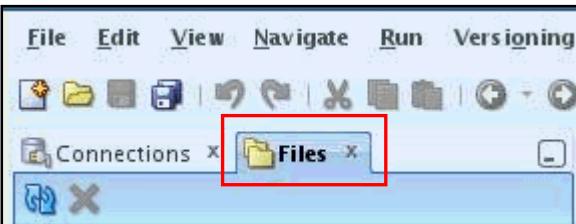
1. Execute the `correlation_setup.sh` script from the `$HOME/labs/solutions/Common_Mistakes` directory. You can find the scripts for all the following cases in your `$HOME/labs/solutions/Common_Mistakes` directory.
2. Analyze a correlated subquery in SQL Developer.
 - a. Go to Tools > Preferences.
 - b. Click Database > Worksheet.
 - c. Select `/home/oracle/labs/solutions/Common_Mistakes` as the default path.



- d. Click OK.
- e. Add the Files tab to the navigator pane. Select **View > Files**.

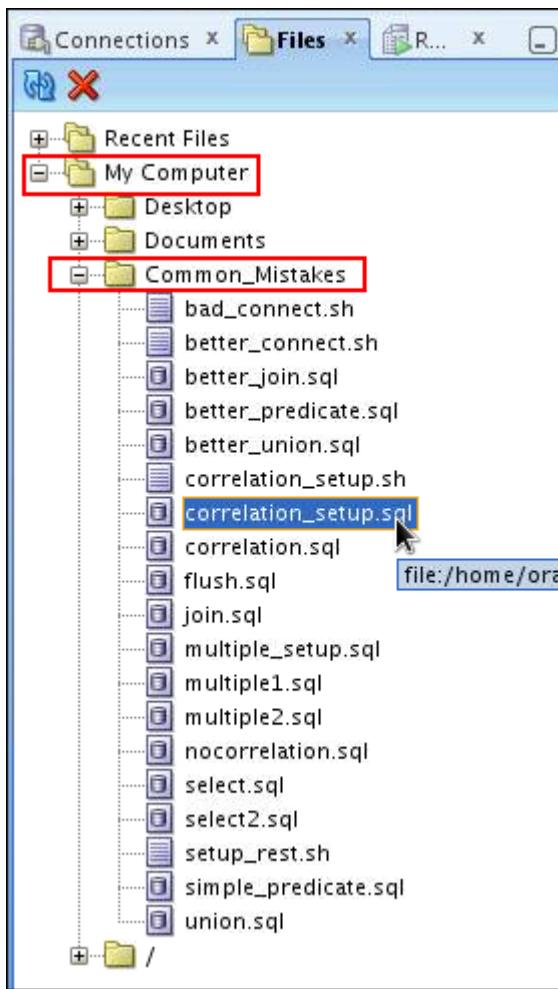


f. Click the Files tab.



g. In the file explorer, browse for the file under

My Computer > Common_Mistakes > correlation_setup.sql



- h. Run the script by clicking the Run Script icon. (Alternatively, press F5.) When you execute the script, you must select a connection. Choose sys_connection from the drop-down menu.

```
grant dba to sh;
```

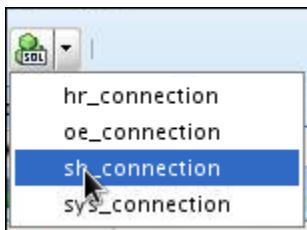
Script Output

Task completed in 0.054 seconds

grant succeeded.

3. Open a SQL Worksheet as the SH user (stay connected to that session until the end of this case).

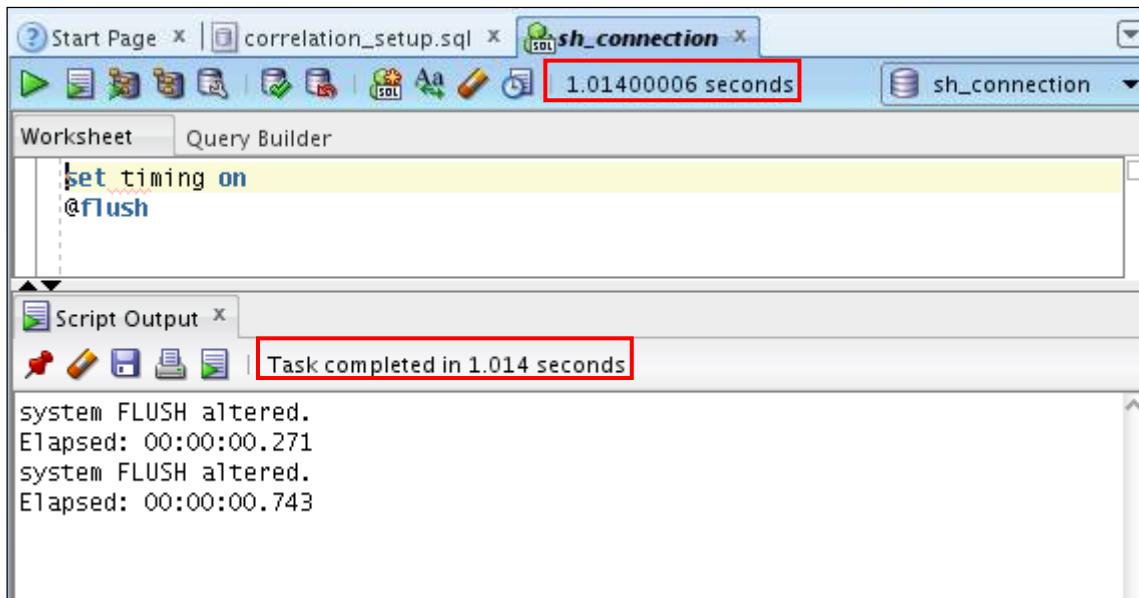
Click the SQL Worksheet button and select sh_connection from the drop-down list.



4. Execute the following command:

```
set timing on  
@flush
```

The goal of the first command is to tell you how long any command take to execute. The flush.sql script flushes both the shared pool and the buffer cache to avoid most caching effects so that you can make good comparisons between two executions. **Note:** You should *not* use commands found in this script on a production system.



Note: SQL Developer displays the timing information in the worksheet by default and in the output pane as a result of the SET TIMING ON command.

5. From the same session, execute the following statement and note the time it takes to execute. (You can use the correlation.sql script.)

```
SELECT COUNT(*)  
FROM products p  
WHERE prod_list_price < 1.15 * (SELECT avg(unit_cost)  
                                FROM costs c  
                                WHERE c.prod_id = p.prod_id);
```

The screenshot shows the Oracle SQL Developer interface. In the top tab bar, there are three tabs: 'correlation_setup.sql', 'sh_connection', and 'correlation.sql'. The 'correlation.sql' tab is active. Below the tabs is a toolbar with various icons. A red box highlights the status bar at the bottom of the toolbar, which displays '0.48300001 seconds'. The main workspace contains a SQL Worksheet with the following query:

```
SELECT COUNT(*)
FROM products p
WHERE prod_list_price < 1.15 * (SELECT avg(unit_cost)
                                FROM costs c
                                WHERE c.prod_id = p.prod_id);
```

Below the worksheet is a 'Script Output' window. A red box highlights the message 'Task completed in 0.483 seconds' in the output. The output itself shows the result of the COUNT(*) query:

```
COUNT(*)
-----
46
```

At the bottom of the output window, it says 'Elapsed: 00:00:00.483'.

6. Before trying to fix the previous statement, flush your environment again by using the `flush.sql` script from the SQL Developer `sh_connection` session.

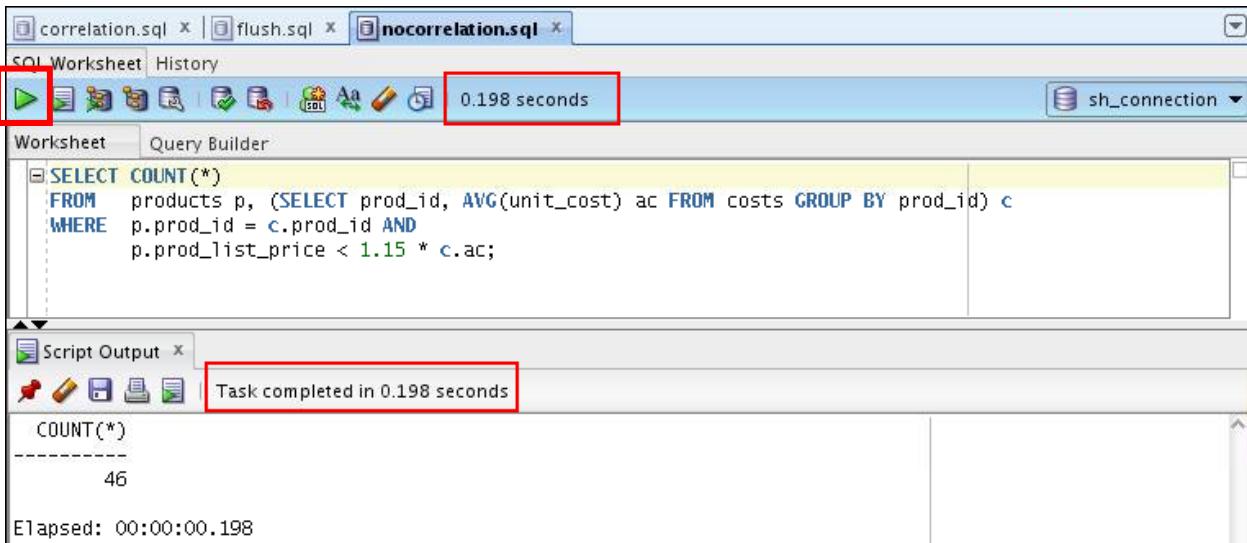
The screenshot shows the Oracle SQL Developer interface with a new tab 'flush.sql' active. The status bar at the bottom of the toolbar shows '1.398 seconds'. The 'Worksheet' tab is selected, displaying the following commands:

```
alter system flush shared_pool;
alter system flush buffer_cache;
```

Below the worksheet is a 'Script Output' window. A red box highlights the message 'Task completed in 1.398 seconds' in the output. The output shows two 'system FLUSH altered.' messages and their respective elapsed times.

```
system FLUSH altered.
Elapsed: 00:00:00.108
system FLUSH altered.
Elapsed: 00:00:01.290
```

7. How do you rewrite this statement to enhance performance? Test your solution. You should discuss this with your instructor. (Alternatively, execute `nocorrelation.sql`.)



8. The `setup_rest.sh` script was executed by the `oracle` user as part of the class setup to set up the environment for all the examples that follow. The `setup_rest.sh` script is listed here. You can find the scripts for all the following cases in your `/home/oracle/labs/solutions/Common_Mistakes` directory.

```
#!/bin/bash

cd /home/oracle/labs/solutions/Common_Mistakes

sqlplus / as sysdba <<EOF

set echo on
drop user cm cascade;

create user cm identified by cm default tablespace users
temporary tablespace temp;

grant connect, resource, dba to cm;

connect cm/cm

drop table orders purge;

create table orders (order_id char(50) primary key,
order_total number, customer_name varchar2(300));

begin
for i in 1..500000 loop
```



```
insert into job_history
values(mod(i,1000) , 'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
end loop;
commit;
end;
/

create index job_history_empid_indx on job_history(employee_id);

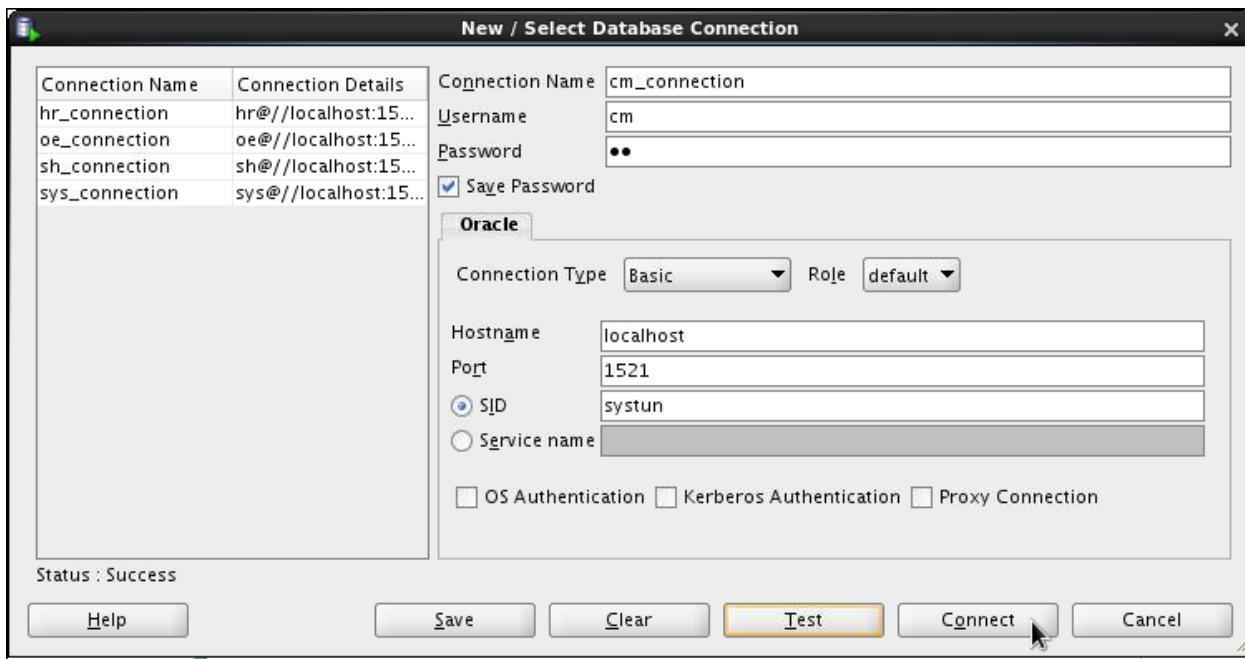
drop table old purge;
drop table new purge;

create table old(name varchar2(10), other varchar2(500));
create table new(name varchar2(10), other varchar2(500));

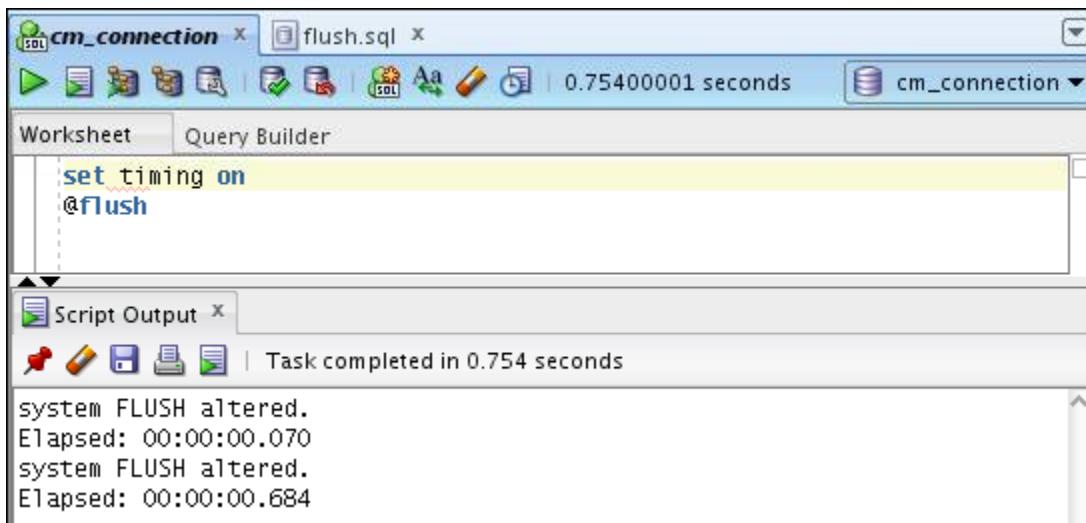
begin
for i in 1..500000 loop
insert into old
values(i, 'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa');
end loop;
commit;
end;
/

begin
for i in 1..500000 loop
insert into new
values(500000+i, 'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa');
end loop;
commit;
end;
/
```

9. Create and connect to the CM user. Stay connected in that session until further notice.



10. Set timings on and flush your environment again before starting the second case. You can use the `set timing on` command and the `flush.sql` script for this.



Case 2: Join Conditions

11. The second case that you analyze is a join case. Using the `cm_connection`, open and execute the `join.sql` script shown here. Note the time it takes to complete:

```
SELECT count(*)
FROM   job_history jh, employees e
WHERE  substr(to_char(e.employee_id),1) =
substr(to_char(jh.employee_id),1);
```

A screenshot of the Oracle SQL Developer interface. The top window is titled 'cm_connection' and shows three tabs: 'join.sql' (selected), 'flush.sql', and 'History'. The status bar at the bottom of this window displays '12.5989998 seconds'. Below this is the 'Worksheet' tab where a SQL query is written:

```
SELECT count(*)
FROM job_history jh, employees e
WHERE substr(to_char(e.employee_id),1) = substr(to_char(jh.employee_id),1);
```

The bottom window is titled 'Script Output' and shows the results of the query. A message 'Task completed in 12.599 seconds' is highlighted with a red box. The output shows:

```
COUNT(*)
-----
499500
```

Elapsed: 00:00:12.599

12. Before trying to fix the previous statement, flush your environment again by using the `flush.sql` script from your SQL Developer `cm_connection` session.

A screenshot of the Oracle SQL Developer interface. The top window is titled 'cm_connection' and shows three tabs: 'join.sql', 'flush.sql' (selected), and 'History'. The status bar at the bottom of this window displays '0.65200001 seconds'. Below this is the 'Worksheet' tab where the following command is written:

```
alter system flush shared_pool;
alter system flush buffer_cache;
```

The bottom window is titled 'Script Output' and shows the results of the command. A message 'Task completed in 0.652 seconds' is highlighted with a red box. The output shows:

```
system FLUSH altered.
Elapsed: 00:00:00.045
system FLUSH altered.
Elapsed: 00:00:00.607
```

13. How would you rewrite the previous query for better performance? Test your solution. You should discuss this with your instructor. (Alternatively, open and execute the `better_join.sql` script.) Why is this query so much faster?

```

SELECT count(*)
FROM   job_history jh, employees e
WHERE  e.employee_id = jh.employee_id;

```

Task completed in 2.05 seconds

COUNT(*)

499500

Elapsed: 00:00:02.050

- The function on the join conditions in the first query prevents the use of a normal index. Removing the functions allows the indexes to be used.
14. Before analyzing the third case, flush your environment again by using the `flush.sql` script from your SQL Developer `cm_connection` session.

```

alter system flush shared_pool;
alter system flush buffer_cache;

```

Task completed in 4.114 seconds

system FLUSH altered.
Elapsed: 00:00:00.045
system FLUSH altered.
Elapsed: 00:00:00.607

Case 3: Simple Predicate

15. The third case you analyze is a simple predicate case. Still using `cm_connection` from your SQL Developer session, execute the following query (alternatively, open and execute the `simple_predicate.sql` script) and note the time it takes to complete:

```
SELECT * FROM orders WHERE order_id_char = 1205;
```

The screenshot shows the Oracle SQL Worksheet interface. In the top panel, there is a toolbar with various icons and a status bar indicating "4.89499998 seconds". Below the toolbar, the "Worksheet" tab is selected. A query is entered in the worksheet pane: `SELECT * FROM orders WHERE order_id_char = 1205;`. In the "Script Output" pane at the bottom, the message "Task completed in 4.895 seconds" is displayed. A red box highlights this message. Below it, the results of the query are shown in a table:

| ORDER_ID_CHAR | ORDER_TOTAL | CUSTOMER_NAME |
|---------------|-------------|----------------------------|
| 1205 | 100 | aaaaaaaaaaaaaaaaaaaaaaaaaa |

Elapsed: 00:00:04.895

16. Before trying to fix the SELECT statement in step 20, flush your environment again using the flush.sql script.

The screenshot shows the Oracle SQL Worksheet interface. In the top panel, there is a toolbar with various icons and a status bar indicating "4.11399984 s". Below the toolbar, the "Worksheet" tab is selected. A script is entered in the worksheet pane: `alter system flush shared_pool;
alter system flush buffer_cache;`. In the "Script Output" pane at the bottom, the message "Task completed in 4.114 seconds" is displayed. A red box highlights this message. Below it, the output of the flush command is shown:

```
system FLUSH altered.  
Elapsed: 00:00:00.045  
system FLUSH altered.  
Elapsed: 00:00:00.607
```

17. How would you rewrite the previous statement for better performance? Test your solution. You should discuss this with your instructor. (Alternatively, open and execute the better_predicate.sql.) Why does this query execute more quickly?

```
SELECT * FROM orders WHERE order_id_char = '1205';
```

The screenshot shows the Oracle SQL Developer interface. In the top tab bar, there are three tabs: 'cm_connection' (selected), 'flush.sql', and 'better_predicate.sql'. The 'better_predicate.sql' tab is highlighted with a red box. Below the tabs is a toolbar with various icons. The status bar at the bottom of the toolbar displays the text '0.022 seconds'. The main workspace shows a 'Worksheet' tab selected, containing the SQL query: 'SELECT * FROM orders WHERE order_id_char = '1205';'. Below the query is a table output area. The table has three columns: 'ORDER_ID_CHAR', 'ORDER_TOTAL', and 'CUSTOMER_NAME'. A single row is present with values '1205', '100', and a long string of 'a's. At the bottom of the table area, it says 'Elapsed: 00:00:00.022'.

- This version runs faster because the data type `order_id_char` column matches the data type of the literal value.
18. Before proceeding with the next analysis, flush your environment again by using the `flush.sql` script.

The screenshot shows the Oracle SQL Developer interface. The top tab bar has 'cm_connection' selected. The main workspace shows a 'Worksheet' tab selected, containing the SQL script: 'alter system flush shared_pool; alter system flush buffer_cache;'. Below the script is a 'Script Output' window. It shows the output: 'system FLUSH altered.' followed by 'Elapsed: 00:00:00.045', then 'system FLUSH altered.' followed by 'Elapsed: 00:00:00.607'. The status bar at the bottom of the toolbar displays the text '4.11399984 s'.

Case 4: Union

19. The fourth case is a UNION case. Execute the following query (alternatively, you can use `union.sql`) and note the time it takes to complete:

```
select count(*)
from (select name from old union select name from new);
```

The screenshot shows the Oracle SQL Worksheet interface. In the top tab bar, there are three tabs: 'cm_connection' (selected), 'flush.sql', and 'union.sql'. The 'union.sql' tab is active. The 'Worksheet' tab is selected in the toolbar. The main workspace contains the following SQL code:

```
select count(*)
from (select name from old union select name from new);
```

In the 'Script Output' pane, the results are displayed:

```
COUNT(*)  
-----  
1000000  
Elapsed: 00:00:08.391
```

A red box highlights the message "Task completed in 8.391 seconds".

20. Before investigating a better solution, flush your environment again by using the `flush.sql` script.

The screenshot shows the Oracle SQL Worksheet interface. The 'Worksheet' tab is selected. The main workspace contains the following SQL code:

```
alter system flush shared_pool;
alter system flush buffer_cache;
```

In the 'Script Output' pane, the results are displayed:

```
system FLUSH altered.
Elapsed: 00:00:00.045
system FLUSH altered.
Elapsed: 00:00:00.607
```

21. How would you rewrite the previous statement for better performance? Test your solution. You should discuss this with your instructor. (Alternatively, open and execute the `better_union.sql` script.) Why does this query execute more quickly?

Note: This query assumes that the `old` and `new` tables are disjoint sets, and there are no duplicate rows.

```
select count(*)
from (select name from old union all select name from new);
```

The screenshot shows the Oracle SQL Developer interface. In the top tab bar, there are tabs for 'cm_connection' (selected), 'flush.sql', 'union.sql', and 'better_union.sql'. The main area is a 'Worksheet' tab showing the following SQL code:

```
select count(*)  
from (select name from old union all select name from new);
```

Below the worksheet, in the 'Script Output' tab, the results are displayed:

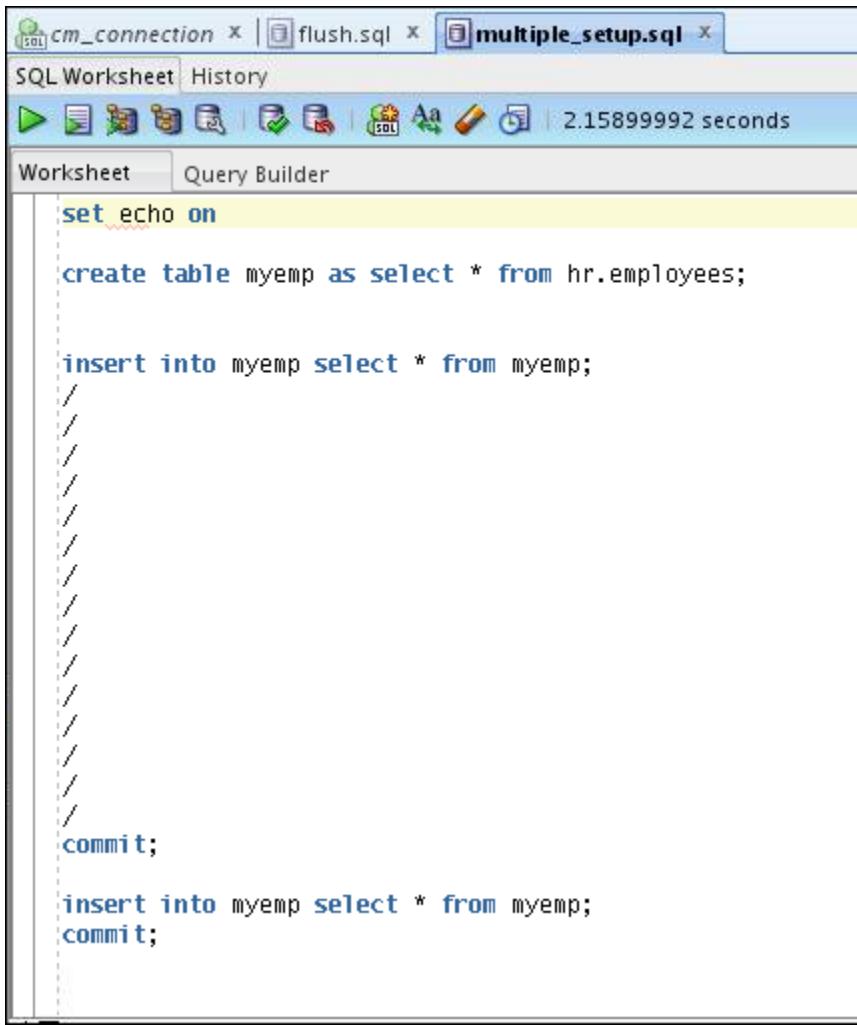
```
COUNT(*)  
-----  
1000000
```

At the bottom of the output window, it says 'Elapsed: 00:00:00.740'. A red box highlights the message 'Task completed in 0.74 seconds'.

- The first query uses a UNION, which requires a sort to remove duplicate rows. The second uses a UNION ALL, which does not remove duplicates. This strategy works well when the tables are disjoint or the duplicate rows can be removed from one of the tables by using a WHERE clause.

Case 5: Combining SQL Statements

22. Execute the `multiple_setup.sql` script the `cm_connection` to set up the environment for this case.



The screenshot shows a SQL Worksheet window titled "multiple_setup.sql". The window has tabs for "Worksheet" and "Query Builder", with "Worksheet" selected. The code area contains the following SQL statements:

```
set echo on
create table myemp as select * from hr.employees;
insert into myemp select * from myemp;
/
/
/
/
/
/
commit;
insert into myemp select * from myemp;
commit;
```

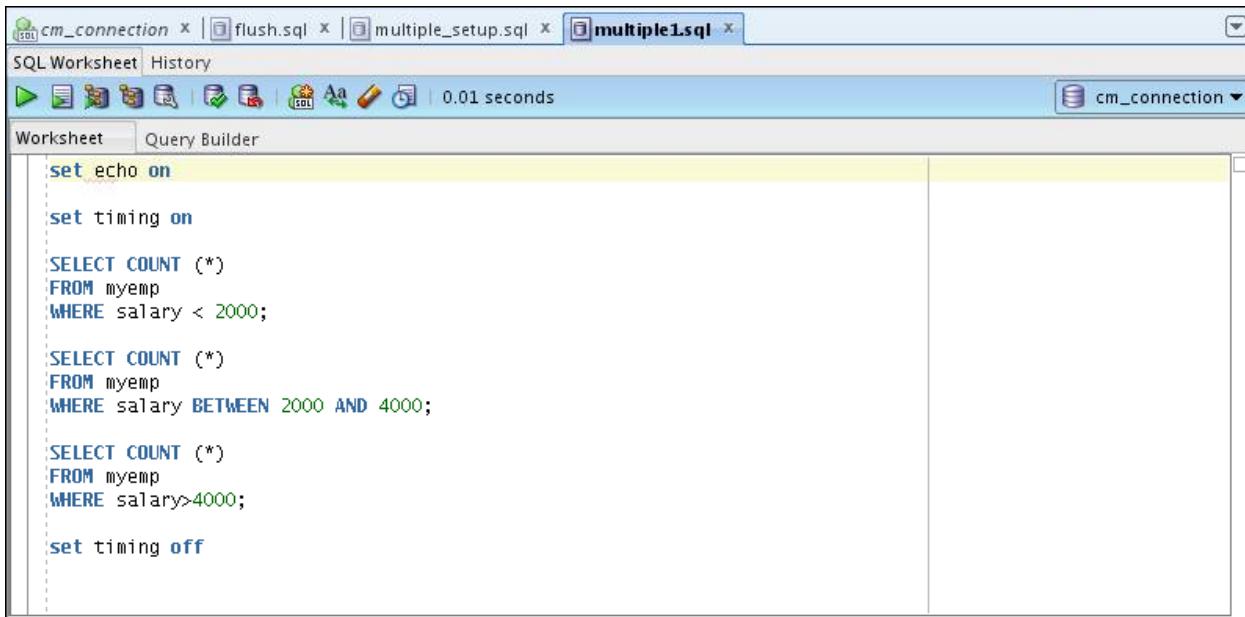
Output:



The screenshot shows the 'Script Output' window of Oracle SQL Developer. The title bar says 'Script Output'. Below it, there are icons for save, print, and copy. A message 'Task completed in 2.159 seconds' is displayed. The main area contains the following SQL commands and their execution details:

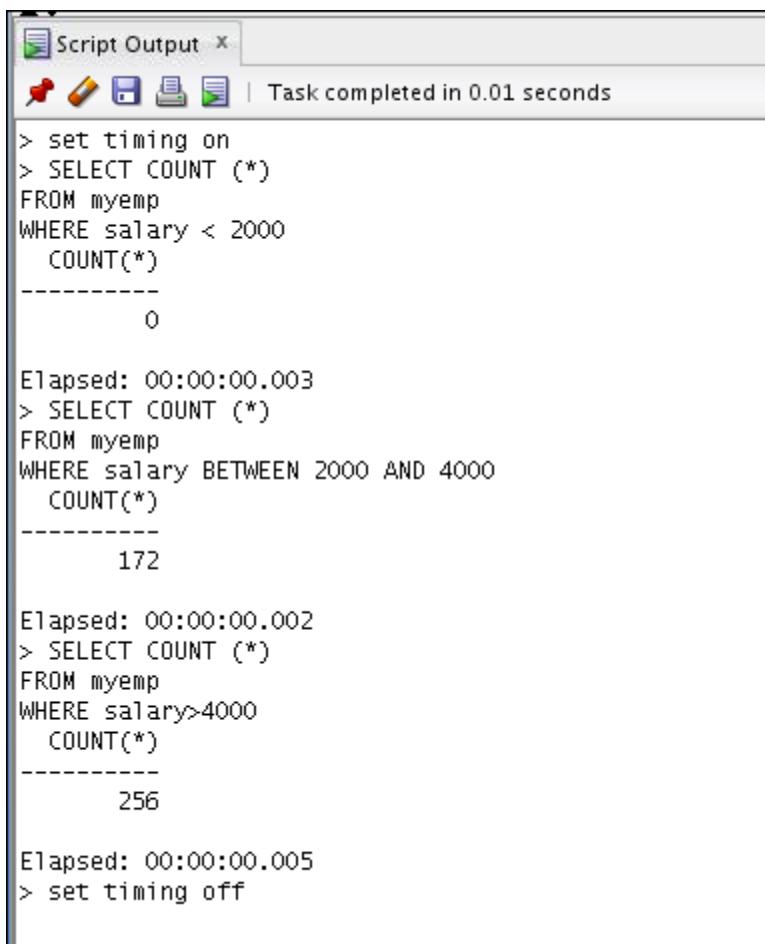
```
> create table myemp as select * from hr.employees
Error starting at line 2 in command:
create table myemp as select * from hr.employees
Error report:
SQL Command: table MYEMP
Failed: Warning: execution completed with warning
Elapsed: 00:00:02.051
> insert into myemp select * from myemp
107 rows inserted.
Elapsed: 00:00:00.015
> /
> /
> /
> /
> /
> /
> /
> /
> /
> /
> /
> /
> /
> /
> /
> /
> /
> /
> /
> /
> commit
committed.
Elapsed: 00:00:00.069
> insert into myemp select * from myemp
214 rows inserted.
Elapsed: 00:00:00.011
> commit
committed.
Elapsed: 00:00:00.013
```

23. Open and execute the `multiple1.sql` script by using the `cm_connection` and note the total time it takes to execute.



The screenshot shows the Oracle SQL Worksheet interface. The title bar has tabs for 'cm_connection', 'flush.sql', 'multiple_setup.sql', and 'multiple1.sql'. The main area is titled 'Worksheet' and contains the following SQL code:

```
set echo on
set timing on
SELECT COUNT (*)
FROM myemp
WHERE salary < 2000;
SELECT COUNT (*)
FROM myemp
WHERE salary BETWEEN 2000 AND 4000;
SELECT COUNT (*)
FROM myemp
WHERE salary>4000;
set timing off
```

Output:

The screenshot shows the 'Script Output' window. It displays the SQL commands run and their results. The output is as follows:

```
Script Output x
Task completed in 0.01 seconds
> set timing on
> SELECT COUNT (*)
FROM myemp
WHERE salary < 2000
COUNT(*)
-----
0

Elapsed: 00:00:00.003
> SELECT COUNT (*)
FROM myemp
WHERE salary BETWEEN 2000 AND 4000
COUNT(*)
-----
172

Elapsed: 00:00:00.002
> SELECT COUNT (*)
FROM myemp
WHERE salary>4000
COUNT(*)
-----
256

Elapsed: 00:00:00.005
> set timing off
```

24. How would you rewrite the statements found in the `multiple1.sql` script for better performance? Use `multiple2.sql` script.

The screenshot shows the Oracle SQL Developer interface. In the top tab bar, the `multiple2.sql` tab is active. The status bar at the top center displays "0.004 seconds". The main workspace contains the following SQL code:

```

set echo on
set timing on
SELECT COUNT (CASE WHEN salary < 2000
                    THEN 1 ELSE null END) count1,
       COUNT (CASE WHEN salary BETWEEN 2001 AND 4000
                    THEN 1 ELSE null END) count2,
       COUNT (CASE WHEN salary > 4000
                    THEN 1 ELSE null END) count3
  FROM myemp;
  
```

In the bottom pane, the `Script Output` window shows the results of the query:

```

> set timing on
> SELECT COUNT (CASE WHEN salary < 2000
                    THEN 1 ELSE null END) count1,
       COUNT (CASE WHEN salary BETWEEN 2001 AND 4000
                    THEN 1 ELSE null END) count2,
       COUNT (CASE WHEN salary > 4000
                    THEN 1 ELSE null END) count3
  FROM myemp
  COUNT1      COUNT2      COUNT3
  -----      -----
        0          172         256
Elapsed: 00:00:00.004
  
```

The message "Task completed in 0.004 seconds" is highlighted with a red box in the output window.

- A single SQL statement will usually execute more quickly than multiple statements.

Case 6: Database Connection Management

25. You now analyze a sixth case that deals with database connections. Execute the `bad_connect.sh` script from your terminal window connected as the `oracle` user. Note the time it takes to complete.

```
$ cd /home/oracle/labs/solutions/Common_Mistakes
$ ./bad_connect.sh
```

```
#!/bin/bash

cd /home/oracle/labs/solutions/Common_Mistakes

STREAM_NUM=0
```

```
MAX_STREAM=500

date

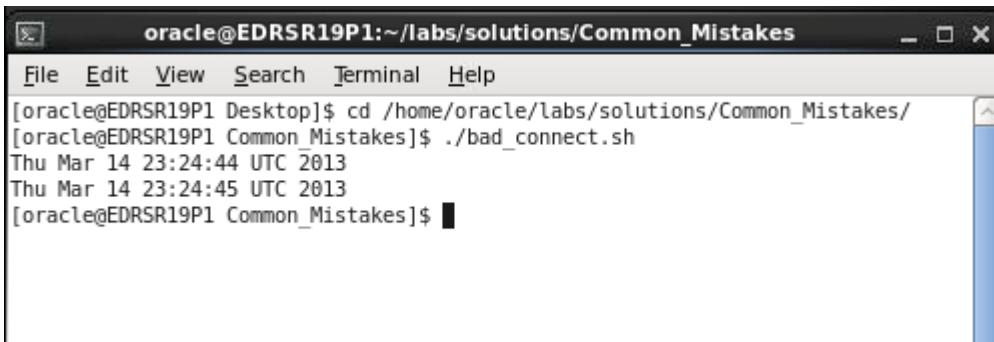
while [ $STREAM_NUM -lt $MAX_STREAM ] ; do

    # one more
    let STREAM_NUM=$STREAM_NUM+1

    # start one more stream
    sqlplus -s cm/cm @select.sql >> /tmp/bad_connect.log 2>&1

done

date
```



A screenshot of a terminal window titled "oracle@EDRSR19P1:~/labs/solutions/Common Mistakes". The window shows the command line history:

```
[oracle@EDRSR19P1 Desktop]$ cd /home/oracle/labs/solutions/Common Mistakes/
[oracle@EDRSR19P1 Common Mistakes]$ ./bad_connect.sh
Thu Mar 14 23:24:44 UTC 2013
Thu Mar 14 23:24:45 UTC 2013
[oracle@EDRSR19P1 Common Mistakes]$
```

26. Analyze the `bad_connect.sh` script and try to find a better solution to enhance the performance of that application. Test your solution. You should discuss this with your instructor. Run the `./better_connect.sh` script.

```
$ cd /home/oracle/labs/solutions/Common Mistakes
$ ./better_connect.sh
```

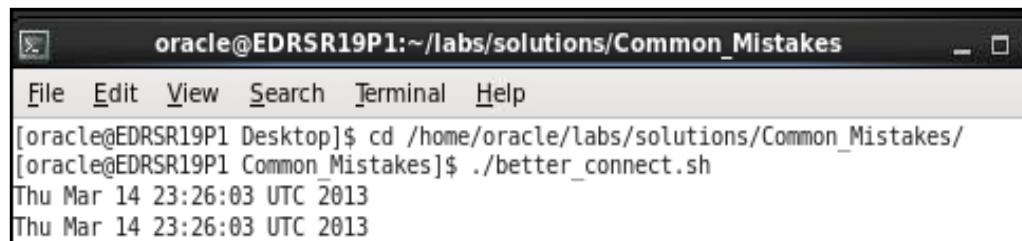
```
#!/bin/bash

cd /home/oracle/labs/solutions/Common Mistakes

date

sqlplus -s cm/cm @select2.sql >> /tmp/better_connect.log 2>&1

date
```



A screenshot of a terminal window titled "oracle@EDRSR19P1:~/labs/solutions/Common_Mistakes". The window shows a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". Below the menu is a command-line session:
[oracle@EDRSR19P1 Desktop]\$ cd /home/oracle/labs/solutions/Common_Mistakes/
[oracle@EDRSR19P1 Common_Mistakes]\$./better_connect.sh
Thu Mar 14 23:26:03 UTC 2013
Thu Mar 14 23:26:03 UTC 2013

- The first script executes a SQL 500 times in a loop that creates 500 connections and executes the SQL statement once in each. The second script creates one connection, and then executes the same SQL statement 500 times.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Error : You are not a Valid Partner use only

Practices for Lesson 5: Optimizer Fundamentals

Chapter 5

Practices for Lesson 5: Overview

Practices Overview

In these practices, you will create an event 10053 trace file, and then review the sections to recognize the contents of this file. You also open and analyze a SQL trace file by using SQL Developer. It is an optional practice.

Note: This practice is only for demonstration purposes. Do *not* use it on your production system unless explicitly asked to by Oracle Support Services.

Practice 5-1: Understanding Optimizer Decisions (Optional)

Overview

In this practice, you analyze optimizer decisions related to which execution plan to use. All the scripts needed for this practice can be found in your \$HOME/labs/solutions/Trace_Event directory.

Tasks

1. Execute the te_setup.sh script. This script executes the following query and generates a trace file that contains all optimizer decisions:

```
SELECT ch.channel_class, c.cust_city, t.calendar_quarter_desc,
SUM(s.amount_sold) sales_amount
FROM sh.sales s,sh.times t,sh.customers c,sh.channels ch
WHERE s.time_id = t.time_id AND
      s.cust_id = c.cust_id AND
      s.channel_id = ch.channel_id AND
      c.cust_state_province = 'CA' AND
      ch.channel_desc IN ('Internet','Catalog') AND
      t.calendar_quarter_desc IN ('1999-Q1','1999-Q2')
GROUP BY ch.channel_class, c.cust_city, t.calendar_quarter_desc;

$ cd /home/oracle/labs/solutions/Trace_Event
$ ./te_setup.sh
```

2. With the help of your instructor, examine the generated trace file and interpret the important parts of the trace file.

The 10053 trace file output is broken down into a number of sections that broadly reflect the stages that the optimizer goes through in evaluating a plan. These stages are as follows: query, parameters used by the optimizer, base statistical information, base table access cost, join order and method computations, and recosting for special features, such as query transformations.

Note: The output can be different based on your practice environment.

```
$ cd /home/oracle/labs/solutions/Trace_Event
$ cat myoptimizer.trc
```

Output:

```
Trace file
/u01/app/oracle/diag/rdbms/systun/systun/trace/systun_ora_31520_
MYOPTIMIZER.trc
Oracle Database 12c Enterprise Edition Release 12.1.0 -
Production
With the Partitioning, OLAP, Data Mining and Real Application
Testing options
```

```
ORACLE_HOME = /u01/app/oracle/product/12.1.0/dbhome_1
System name: Linux
Node name: EDRSR19P1
Release: 2.6.18-164.el5
Version: #1 SMP Thu Sep 3 02:16:47 EDT 2009
Machine: i686
Instance name: systun
Redo thread mounted by this instance: 1
Oracle process number: 39
Unix process pid: 28709, image: oracle@EDRSR19P1 (TNS V1-V3)

*** 2013-06-14 07:51:05.353
*** SESSION ID:(48.19088) 2013-06-14 07:51:05.353
*** CLIENT ID:() 2010-08-04 07:51:05.353
*** SERVICE NAME:(SYS$USERS) 2013-06-14 07:51:05.353
*** MODULE NAME:(sqlplus@EDRSR19P1 (TNS V1-V3)) 2013-06-14
07:51:05.353
*** ACTION NAME:() 2013-06-14 07:51:05.353

Registered qb: SEL$1 0x5c3a04 (PARSER)
-----
QUERY BLOCK SIGNATURE
-----
signature (): qb_name=SEL$1 nbfros=4 flg=0
fro(0): flg=4 objn=74136 hint_alias="C"@"SEL$1"
fro(1): flg=4 objn=74132 hint_alias="CH"@"SEL$1"
fro(2): flg=4 objn=74068 hint_alias="S"@"SEL$1"
fro(3): flg=4 objn=74126 hint_alias="T"@"SEL$1"

*** 2013-06-14 07:51:05.363
SPM: statement not found in SMB

*****
Automatic degree of parallelism (ADOP)
*****
Automatic degree of parallelism is disabled: Parameter.

PM: Considering predicate move-around in query block SEL$1 (#0)
*****
Predicate Move-Around (PM)
*****
```

OPTIMIZER INFORMATION

```
*****
----- Current SQL Statement for this session
(sql_id=70fqjd9u1zk7c) -----
SELECT ch.channel_class, c.cust_city, t.calendar_quarter_desc,
SUM(s.amount_sold) sales_amount
FROM sh.sales s,sh.times t,sh.customers c,sh.channels ch
WHERE s.time_id = t.time_id AND
s.cust_id = c.cust_id AND
s.channel_id = ch.channel_id AND
c.cust_state_province = 'CA' AND
ch.channel_desc IN ('Internet','Catalog') AND
t.calendar_quarter_desc IN ('1999-Q1','1999-Q2')
GROUP BY ch.channel_class, c.cust_city, t.calendar_quarter_desc
*****
```

Legend

The following abbreviations are used by optimizer trace.

CBQT - cost-based query transformation

JPPD - join predicate push-down

OJPPD - old-style (non-cost-based) JPPD

FPD - filter push-down

PM - predicate move-around

CVM - complex view merging

SPJ - select-project-join

SJC - set join conversion

SU - subquery unnesting

OBYE - order by elimination

OST - old style star transformation

ST - new (cbqt) star transformation

CNT - count(col) to count(*) transformation

JE - Join Elimination

JF - join factorization

SLP - select list pruning

DP - distinct placement

qb - query block

LB - leaf blocks

DK - distinct keys

LB/K - average number of leaf blocks per key

DB/K - average number of data blocks per key

CLUF - clustering factor

NDV - number of distinct values

```
Resp - response cost
Card - cardinality
Resc - resource cost
NL - nested loops (join)
SM - sort merge (join)
HA - hash (join)
CPUSPEED - CPU Speed
IOTFRSPEED - I/O transfer speed
IOSEEKTIM - I/O seek time
SREADTIM - average single block read time
MREADTIM - average multiblock read time
MBRC - average multiblock read count
MAXTHR - maximum I/O system throughput
SLAVETHR - average slave I/O throughput
dmeth - distribution method
  1: no partitioning required
  2: value partitioned
  4: right is random (round-robin)
 128: left is random (round-robin)
   8: broadcast right and partition left
  16: broadcast left and partition right
  32: partition left using partitioning of right
  64: partition right using partitioning of left
 256: run the join in serial
   0: invalid distribution method
sel - selectivity
ptn - partition
*****
PARAMETERS USED BY THE OPTIMIZER
*****
*****
PARAMETERS WITH ALTERED VALUES
*****
Compilation Environment Dump
_smm_min_size          = 286 KB
_smm_max_size           = 57344 KB
_smm_px_max_size        = 143360 KB
Bug Fix Control Environment

*****
PARAMETERS WITH DEFAULT VALUES
```

```
*****
Compilation Environment Dump
optimizer_mode_hinted          = false
optimizer_features_hinted       = 0.0.0
parallel_execution_enabled      = true
parallel_query_forced_dop      = 0
parallel_dml_forced_dop        = 0
parallel_ddl_forced_degree     = 0
parallel_ddl_forced_instances  = 0
_query_rewrite_fudge           = 90
optimizer_features_enable       = 12.1.0.1
_optimizer_search_limit         = 5
cpu_count                       = 1
...
parallel_hinted                = none
_sql_compatibility              = 0
_optimizer_use_feedback          = true
_optimizer_try_st_before_jppd   = true
Bug Fix Control Environment
fix 3834770 = 1
fix 3746511 = enabled
fix 4519016 = enabled
fix 3118776 = enabled
fix 4488689 = enabled
fix 2194204 = disabled
...
*****  

PARAMETERS IN OPT_PARAM HINT
*****
*****  

Column Usage Monitoring is ON: tracking level = 1
*****  

Considering Query Transformations on query block SEL$1 (#0)
*****
Query transformations (QT)
*****
CBQT: Validity checks passed for 70fqjd9u1zk7c.
CSE: Considering common sub-expression elimination in query
block SEL$1 (#0)
*****
```

```

Common Subexpression elimination (CSE)
*****
CSE:      CSE not performed on query block SEL$1 (#0).
OBYE:     Considering Order-by Elimination from view SEL$1 (#0)
*****
Order-by elimination (OBYE)
*****
OBYE:     OBYE bypassed: no order by to eliminate.
JE:      Considering Join Elimination on query block SEL$1 (#0)
*****
Join Elimination (JE)
*****
SQL:***** UNPARSED QUERY IS *****
SELECT "CH"."CHANNEL_CLASS" "CHANNEL_CLASS", "C"."CUST_CITY"
"CUST_CITY", "T"."CALENDAR_QUARTER_DESC"
"CALENDAR_QUARTER_DESC", SUM("S"."AMOUNT_SOLD") "SALES_AMOUNT"
FROM "SH"."SALES" "S", "SH"."TIMES" "T", "SH"."CUSTOMERS"
"C", "SH"."CHANNELS" "CH" WHERE "S"."TIME_ID"="T"."TIME_ID" AND
"S"."CUST_ID"="C"."CUST_ID" AND
"S"."CHANNEL_ID"="CH"."CHANNEL_ID" AND
"C"."CUST_STATE_PROVINCE"='CA' AND
("CH"."CHANNEL_DESC"='Internet' OR
"CH"."CHANNEL_DESC"='Catalog') AND
("T"."CALENDAR_QUARTER_DESC"='1999-Q1' OR
"T"."CALENDAR_QUARTER_DESC"='1999-Q2') GROUP BY
"CH"."CHANNEL_CLASS", "C"."CUST_CITY", "T"."CALENDAR_QUARTER_DESC"
SQL:***** UNPARSED QUERY IS *****
SELECT "CH"."CHANNEL_CLASS" "CHANNEL_CLASS", "C"."CUST_CITY"
"CUST_CITY", "T"."CALENDAR_QUARTER_DESC"
"CALENDAR_QUARTER_DESC", SUM("S"."AMOUNT_SOLD") "SALES_AMOUNT"
FROM "SH"."SALES" "S", "SH"."TIMES" "T", "SH"."CUSTOMERS"
"C", "SH"."CHANNELS" "CH" WHERE "S"."TIME_ID"="T"."TIME_ID" AND
"S"."CUST_ID"="C"."CUST_ID" AND
"S"."CHANNEL_ID"="CH"."CHANNEL_ID" AND
"C"."CUST_STATE_PROVINCE"='CA' AND
("CH"."CHANNEL_DESC"='Internet' OR
"CH"."CHANNEL_DESC"='Catalog') AND
("T"."CALENDAR_QUARTER_DESC"='1999-Q1' OR
"T"."CALENDAR_QUARTER_DESC"='1999-Q2') GROUP BY
"CH"."CHANNEL_CLASS", "C"."CUST_CITY", "T"."CALENDAR_QUARTER_DESC"
Query block SEL$1 (#0) unchanged
CNT:     Considering count(col) to count(*) on query block SEL$1
(#0)
*****
Count(col) to Count(*) (CNT)
*****
CNT:     COUNT() to COUNT(*) not done.

```

```

JE: Considering Join Elimination on query block SEL$1 (#0)
*****
Join Elimination (JE)
*****
SQL:***** UNPARSED QUERY IS *****

SELECT "CH"."CHANNEL_CLASS" "CHANNEL_CLASS", "C"."CUST_CITY"
"CUST_CITY", "T"."CALENDAR_QUARTER_DESC"
"CALENDAR_QUARTER_DESC", SUM("S".AMOUNT_SOLD) "SALES_AMOUNT"
FROM "SH"."SALES" "S", "SH"."TIMES" "T", "SH"."CUSTOMERS"
"C", "SH"."CHANNELS" "CH" WHERE "S".TIME_ID="T".TIME_ID AND
"S".CUST_ID="C".CUST_ID AND
"S".CHANNEL_ID="CH".CHANNEL_ID AND
"C".CUST_STATE_PROVINCE='CA' AND
("CH".CHANNEL_DESC='Internet' OR
"CH".CHANNEL_DESC='Catalog') AND
("T".CALENDAR_QUARTER_DESC='1999-Q1' OR
"T".CALENDAR_QUARTER_DESC='1999-Q2') GROUP BY
"CH".CHANNEL_CLASS", "C".CUST_CITY", "T".CALENDAR_QUARTER_DESC"
SQL:***** UNPARSED QUERY IS *****

SELECT "CH"."CHANNEL_CLASS" "CHANNEL_CLASS", "C"."CUST_CITY"
"CUST_CITY", "T"."CALENDAR_QUARTER_DESC"
"CALENDAR_QUARTER_DESC", SUM("S".AMOUNT_SOLD) "SALES_AMOUNT"
FROM "SH"."SALES" "S", "SH"."TIMES" "T", "SH"."CUSTOMERS"
"C", "SH"."CHANNELS" "CH" WHERE "S".TIME_ID="T".TIME_ID AND
"S".CUST_ID="C".CUST_ID AND
"S".CHANNEL_ID="CH".CHANNEL_ID AND
"C".CUST_STATE_PROVINCE='CA' AND
("CH".CHANNEL_DESC='Internet' OR
"CH".CHANNEL_DESC='Catalog') AND
("T".CALENDAR_QUARTER_DESC='1999-Q1' OR
"T".CALENDAR_QUARTER_DESC='1999-Q2') GROUP BY
"CH".CHANNEL_CLASS", "C".CUST_CITY", "T".CALENDAR_QUARTER_DESC"
Query block SEL$1 (#0) unchanged
query block SEL$1 (#0) unchanged
Considering Query Transformations on query block SEL$1 (#0)
*****
Query transformations (QT)
*****
CSE: Considering common sub-expression elimination in query
block SEL$1 (#0)
*****
Common Subexpression elimination (CSE)
*****
CSE: CSE not performed on query block SEL$1 (#0).
query block SEL$1 (#0) unchanged
apadrv-start sqlid=8087006336042125548
:

```

```
call(in-use=62772, alloc=81864), compile(in-use=73112,
alloc=77156), execution(in-use=3504, alloc=4060)

*****
Peeked values of the binds in SQL statement
*****

CBQT: Considering cost-based transformation on query block SEL$1
(#0)
*****
COST-BASED QUERY TRANSFORMATIONS
*****
FPD: Considering simple filter push (pre rewrite) in query block
SEL$1 (#0)
FPD: Current where clause predicates
"S"."TIME_ID"="T"."TIME_ID" AND "S"."CUST_ID"="C"."CUST_ID" AND
"S"."CHANNEL_ID"="CH"."CHANNEL_ID" AND
"C"."CUST_STATE_PROVINCE"='CA' AND
("CH"."CHANNEL_DESC"='Internet' OR
"CH"."CHANNEL_DESC"='Catalog') AND
("T"."CALENDAR_QUARTER_DESC"='1999-Q1' OR "T"."C

OBYE: Considering Order-by Elimination from view SEL$1 (#0)
*****
Order-by elimination (OBYE)
*****
OBYE: OBYE bypassed: no order by to eliminate.
Considering Query Transformations on query block SEL$1 (#0)
*****
Query transformations (QT)
*****
CSE: Considering common sub-expression elimination in query
block SEL$1 (#0)
*****
Common Subexpression elimination (CSE)
*****
CSE: CSE not performed on query block SEL$1 (#0).
kkqctdrvTD-start on query block SEL$1 (#0)
kkqctdrvTD-start: :
    call(in-use=62772, alloc=81864), compile(in-use=114972,
alloc=117532), execution(in-use=3504, alloc=4060)

kkqctdrvTD-cleanup: transform(in-use=0, alloc=0) :
    call(in-use=62772, alloc=81864), compile(in-use=115560,
alloc=117532), execution(in-use=3504, alloc=4060)
```

```
kkqctdrvTD-end:  
    call(in-use=62772, alloc=81864), compile(in-use=115976,  
alloc=117532), execution(in-use=3504, alloc=4060)  
  
SJC: Considering set-join conversion in query block SEL$1 (#1)  
*****  
Set-Join Conversion (SJC)  
*****  
SJC: not performed  
CNT: Considering count(col) to count(*) on query block SEL$1  
(#1)  
*****  
Count(col) to Count(*) (CNT)  
*****  
CNT: COUNT() to COUNT(*) not done.  
JE: Considering Join Elimination on query block SEL$1 (#1)  
*****  
Join Elimination (JE)  
*****  
SQL:***** UNPARSED QUERY IS *****  
SELECT "CH"."CHANNEL_CLASS" "CHANNEL_CLASS", "C"."CUST_CITY"  
"CUST_CITY", "T"."CALENDAR_QUARTER_DESC"  
"CALENDAR_QUARTER_DESC", SUM("S"."AMOUNT_SOLD") "SALES_AMOUNT"  
FROM "SH"."SALES" "S", "SH"."TIMES" "T", "SH"."CUSTOMERS"  
"C", "SH"."CHANNELS" "CH" WHERE "S"."TIME_ID"="T"."TIME_ID" AND  
"S"."CUST_ID"="C"."CUST_ID" AND  
"S"."CHANNEL_ID"="CH"."CHANNEL_ID" AND  
"C"."CUST_STATE_PROVINCE"='CA' AND  
("CH"."CHANNEL_DESC"='Internet' OR  
"CH"."CHANNEL_DESC"='Catalog') AND  
("T"."CALENDAR_QUARTER_DESC"='1999-Q1' OR  
"T"."CALENDAR_QUARTER_DESC"='1999-Q2') GROUP BY  
"CH"."CHANNEL_CLASS", "C"."CUST_CITY", "T"."CALENDAR_QUARTER_DESC"  
SQL:***** UNPARSED QUERY IS *****  
SELECT "CH"."CHANNEL_CLASS" "CHANNEL_CLASS", "C"."CUST_CITY"  
"CUST_CITY", "T"."CALENDAR_QUARTER_DESC"  
"CALENDAR_QUARTER_DESC", SUM("S"."AMOUNT_SOLD") "SALES_AMOUNT"  
FROM "SH"."SALES" "S", "SH"."TIMES" "T", "SH"."CUSTOMERS"  
"C", "SH"."CHANNELS" "CH" WHERE "S"."TIME_ID"="T"."TIME_ID" AND  
"S"."CUST_ID"="C"."CUST_ID" AND  
"S"."CHANNEL_ID"="CH"."CHANNEL_ID" AND  
"C"."CUST_STATE_PROVINCE"='CA' AND  
("CH"."CHANNEL_DESC"='Internet' OR  
"CH"."CHANNEL_DESC"='Catalog') AND  
("T"."CALENDAR_QUARTER_DESC"='1999-Q1' OR
```

```
"T". "CALENDAR_QUARTER_DESC"='1999-Q2') GROUP BY
"CH". "CHANNEL_CLASS", "C". "CUST_CITY", "T". "CALENDAR_QUARTER_DESC"
Query block SEL$1 (#1) unchanged
PM: Considering predicate move-around in query block SEL$1 (#1)
*****
Predicate Move-Around (PM)
*****
PM:      PM bypassed: Outer query contains no views.
PM:      PM bypassed: Outer query contains no views.
kkqctdrvTD-start on query block SEL$1 (#1)
kkqctdrvTD-start: :
    call(in-use=82748, alloc=98240), compile(in-use=118884,
alloc=121656), execution(in-use=3504, alloc=4060)

kkqctdrvTD-cleanup: transform(in-use=0, alloc=0) :
    call(in-use=82748, alloc=98240), compile(in-use=119440,
alloc=121656), execution(in-use=3504, alloc=4060)

kkqctdrvTD-end:
    call(in-use=82748, alloc=98240), compile(in-use=119840,
alloc=121656), execution(in-use=3504, alloc=4060)

kkqctdrvTD-start on query block SEL$1 (#1)
kkqctdrvTD-start: :
    call(in-use=82748, alloc=98240), compile(in-use=119840,
alloc=121656), execution(in-use=3504, alloc=4060)

Registered qb: SEL$1 0x610068 (COPY SEL$1)
-----
QUERY BLOCK SIGNATURE
-----
signature(): NULL
*****
Cost-Based Group-By/Distinct Placement
*****
GBP/DP: Checking validity of GBP/DP for query block SEL$1 (#1)
GBP: Checking validity of group-by placement for query block
SEL$1 (#1)
GBP: Bypassed: QB has disjunction.
DP: Checking validity of distinct placement for query block
SEL$1 (#1)
DP: Bypassed: Query has invalid constructs.
kkqctdrvTD-cleanup: transform(in-use=9592, alloc=10596) :
```

```
call(in-use=86820, alloc=98240), compile(in-use=139896,
alloc=146064), execution(in-use=3504, alloc=4060)

kkqctdrvTD-end:
call(in-use=86820, alloc=98240), compile(in-use=129604,
alloc=146064), execution(in-use=3504, alloc=4060)

kkqctdrvTD-start on query block SEL$1 (#1)
kkqctdrvTD-start:
call(in-use=86820, alloc=98240), compile(in-use=129604,
alloc=146064), execution(in-use=3504, alloc=4060)

TE: Checking validity of table expansion for query block SEL$1
(#1)
TE: Bypassed: No relevant table found.
kkqctdrvTD-cleanup: transform(in-use=0, alloc=0) :
call(in-use=87076, alloc=98240), compile(in-use=133212,
alloc=146064), execution(in-use=3504, alloc=4060)

kkqctdrvTD-end:
call(in-use=87076, alloc=98240), compile(in-use=133612,
alloc=146064), execution(in-use=3504, alloc=4060)

TE: Checking validity of table expansion for query block SEL$1
(#1)
TE: Bypassed: No relevant table found.
ST: Query in kkqstardrv:***** UNPARSED QUERY IS *****
SELECT "CH"."CHANNEL_CLASS" "CHANNEL_CLASS", "C"."CUST_CITY"
"CUST_CITY", "T"."CALENDAR_QUARTER_DESC"
"CALENDAR_QUARTER_DESC", SUM("S"."AMOUNT_SOLD") "SALES_AMOUNT"
FROM "SH"."SALES" "S", "SH"."TIMES" "T", "SH"."CUSTOMERS"
"C", "SH"."CHANNELS" "CH" WHERE "S"."TIME_ID"="T"."TIME_ID" AND
"S"."CUST_ID"="C"."CUST_ID" AND
"S"."CHANNEL_ID"="CH"."CHANNEL_ID" AND
"C"."CUST_STATE_PROVINCE"='CA' AND
("CH"."CHANNEL_DESC"='Internet' OR
"CH"."CHANNEL_DESC"='Catalog') AND
("T"."CALENDAR_QUARTER_DESC"='1999-Q1' OR
"T"."CALENDAR_QUARTER_DESC"='1999-Q2') GROUP BY
"CH"."CHANNEL_CLASS", "C"."CUST_CITY", "T"."CALENDAR_QUARTER_DESC"
ST: not valid since star transformation parameter is FALSE
kkqctdrvTD-start on query block SEL$1 (#1)
kkqctdrvTD-start:
call(in-use=87296, alloc=98240), compile(in-use=136524,
alloc=146064), execution(in-use=3504, alloc=4060)
```

```
JF: Checking validity of join factorization for query block
SEL$1 (#1)
JF: Bypassed: not a UNION or UNION-ALL query block.
kkqctdrvTD-cleanup: transform(in-use=0, alloc=0) :
    call(in-use=87296, alloc=98240), compile(in-use=137092,
alloc=146064), execution(in-use=3504, alloc=4060)

kkqctdrvTD-end:
    call(in-use=87296, alloc=98240), compile(in-use=137492,
alloc=146064), execution(in-use=3504, alloc=4060)

JPPD: Considering Cost-based predicate pushdown from query
block SEL$1 (#1)
*****
Cost-based predicate pushdown (JPPD)
*****
kkqctdrvTD-start on query block SEL$1 (#1)
kkqctdrvTD-start: :
    call(in-use=87296, alloc=98240), compile(in-use=137492,
alloc=146064), execution(in-use=3504, alloc=4060)

kkqctdrvTD-cleanup: transform(in-use=0, alloc=0) :
    call(in-use=87296, alloc=98240), compile(in-use=138048,
alloc=146064), execution(in-use=3504, alloc=4060)

kkqctdrvTD-end:
    call(in-use=87296, alloc=98240), compile(in-use=138448,
alloc=146064), execution(in-use=3504, alloc=4060)

JPPD: Applying transformation directives
query block SEL$1 (#1) unchanged
FPD: Considering simple filter push in query block SEL$1 (#1)
"S"."TIME_ID"="T"."TIME_ID" AND "S"."CUST_ID"="C"."CUST_ID" AND
"S"."CHANNEL_ID"="CH"."CHANNEL_ID" AND
"C"."CUST_STATE_PROVINCE"='CA' AND
("CH"."CHANNEL_DESC"='Internet' OR
"CH"."CHANNEL_DESC"='Catalog') AND
("T"."CALENDAR_QUARTER_DESC"='1999-Q1' OR "T"."C
try to generate transitive predicate from check constraints for
query block SEL$1 (#1)
finally: "S"."TIME_ID"="T"."TIME_ID" AND
"S"."CUST_ID"="C"."CUST_ID" AND
"S"."CHANNEL_ID"="CH"."CHANNEL_ID" AND
"C"."CUST_STATE_PROVINCE"='CA' AND
("CH"."CHANNEL_DESC"='Internet' OR
```

```
"CH"."CHANNEL_DESC"='Catalog') AND
("T"."CALENDAR_QUARTER_DESC"='1999-Q1' OR "T"."C

Final query after transformations:***** UNPARSED QUERY IS
*****
SELECT "CH"."CHANNEL_CLASS" "CHANNEL_CLASS", "C"."CUST_CITY"
"CUST_CITY", "T"."CALENDAR_QUARTER_DESC"
"CALENDAR_QUARTER_DESC", SUM("S"."AMOUNT_SOLD") "SALES_AMOUNT"
FROM "SH"."SALES" "S", "SH"."TIMES" "T", "SH"."CUSTOMERS"
"C", "SH"."CHANNELS" "CH" WHERE "S"."TIME_ID"="T"."TIME_ID" AND
"S"."CUST_ID"="C"."CUST_ID" AND
"S"."CHANNEL_ID"="CH"."CHANNEL_ID" AND
"C"."CUST_STATE_PROVINCE"='CA' AND
("CH"."CHANNEL_DESC"='Internet' OR
"CH"."CHANNEL_DESC"='Catalog') AND
("T"."CALENDAR_QUARTER_DESC"='1999-Q1' OR
"T"."CALENDAR_QUARTER_DESC"='1999-Q2') GROUP BY
"CH"."CHANNEL_CLASS", "C"."CUST_CITY", "T"."CALENDAR_QUARTER_DESC"
kkoqbc: optimizing query block SEL$1 (#1)

:
call(in-use=87604, alloc=98240), compile(in-use=138896,
alloc=146064), execution(in-use=3504, alloc=4060)

kkoqbc-subheap (create addr=0x61b144)
*****
QUERY BLOCK TEXT
*****
SELECT ch.channel_class, c.cust_city, t.calendar_quarter_desc,
SUM(s.amount_sold) sales_amount
FROM sh.sales s,sh.times t,sh.customers c,sh.channels ch
WHERE s.time_id = t.time_id AND
s.cust_id = c.cust_id AND
s.channel_id = ch.channel_id
-----
QUERY BLOCK SIGNATURE
-----
signature (optimizer): qb_name=SEL$1 nbffros=4 flg=0
fro(0): flg=0 objn=74136 hint_alias="C"@SEL$1"
fro(1): flg=0 objn=74132 hint_alias="CH"@SEL$1"
fro(2): flg=0 objn=74068 hint_alias="S"@SEL$1"
fro(3): flg=0 objn=74126 hint_alias="T"@SEL$1"

-----
SYSTEM STATISTICS INFORMATION
-----
```

```
Using NOWORKLOAD Stats
CPUSPEEDNW: 2696 millions instructions/sec (default is 100)
IOTFRSPEED: 4096 bytes per millisecond (default is 4096)
IOSEEKTIM: 10 milliseconds (default is 10)
MBRC: -1 blocks (default is 8)

*****
BASE STATISTICAL INFORMATION
*****
Table Stats:::
Table: CHANNELS Alias: CH
#Rows: 5 #Blks: 4 AvgRowLen: 41.00
Index Stats:::
Index: CHANNELS_PK Col#: 1
LVLS: 0 #LB: 1 #DK: 5 LB/K: 1.00 DB/K: 1.00 CLUF: 1.00
*****
Table Stats:::
Table: CUSTOMERS Alias: C
#Rows: 55500 #Blks: 1486 AvgRowLen: 181.00
Index Stats:::
Index: CUSTOMERS_GENDER_BIX Col#: 4
LVLS: 1 #LB: 3 #DK: 2 LB/K: 1.00 DB/K: 2.00 CLUF: 5.00
Index: CUSTOMERS_MARITAL_BIX Col#: 6
LVLS: 1 #LB: 5 #DK: 11 LB/K: 1.00 DB/K: 1.00 CLUF: 18.00
Index: CUSTOMERS_PK Col#: 1
LVLS: 1 #LB: 115 #DK: 55500 LB/K: 1.00 DB/K: 1.00 CLUF: 54405.00
Index: CUSTOMERS_YOB_BIX Col#: 5
LVLS: 1 #LB: 19 #DK: 75 LB/K: 1.00 DB/K: 1.00 CLUF: 75.00
Index: CUST_CUST_CITY_IDX Col#: 9
LVLS: 1 #LB: 161 #DK: 620 LB/K: 1.00 DB/K: 83.00 CLUF: 51600.00
*****
Table Stats:::
Table: TIMES Alias: T
#Rows: 1826 #Blks: 59 AvgRowLen: 198.00
Index Stats:::
Index: TIMES_PK Col#: 1
LVLS: 1 #LB: 5 #DK: 1826 LB/K: 1.00 DB/K: 1.00 CLUF: 53.00
*****
```

```
Table Stats::  
  Table: SALES Alias: S (Using composite stats)  
    #Rows: 918843 #Blks: 1769 AvgRowLen: 29.00  
Index Stats::  
  Index: SALES_CHANNEL_BIX Col#: 4  
    USING COMPOSITE STATS  
    LVLS: 1 #LB: 47 #DK: 4 LB/K: 11.00 DB/K: 23.00 CLUF:  
92.00  
  Index: SALES_CUST_BIX Col#: 2  
    USING COMPOSITE STATS  
    LVLS: 1 #LB: 475 #DK: 7059 LB/K: 1.00 DB/K: 5.00 CLUF:  
35808.00  
  Index: SALES_PROD_BIX Col#: 1  
    USING COMPOSITE STATS  
    LVLS: 1 #LB: 32 #DK: 72 LB/K: 1.00 DB/K: 14.00 CLUF:  
1074.00  
  Index: SALES_PROMO_BIX Col#: 5  
    USING COMPOSITE STATS  
    LVLS: 1 #LB: 30 #DK: 4 LB/K: 7.00 DB/K: 13.00 CLUF:  
54.00  
  Index: SALES_TIME_BIX Col#: 3  
    USING COMPOSITE STATS  
    LVLS: 1 #LB: 59 #DK: 1460 LB/K: 1.00 DB/K: 1.00 CLUF:  
1460.00  
Access path analysis for SALES  
*****  
SINGLE TABLE ACCESS PATH  
  Single Table Cardinality Estimation for SALES[S]  
  Table: SALES Alias: S  
    Card: Original: 918843.000000 Rounded: 918843 Computed:  
918843.00 Non Adjusted: 918843.00  
  Access Path: TableScan  
    Cost: 489.06 Resp: 489.06 Degree: 0  
      Cost_io: 481.00 Cost_cpu: 260685437  
      Resp_io: 481.00 Resp_cpu: 260685437  
***** trying bitmap/domain indexes *****  
  Access Path: index (FullScan)  
    Index: SALES_CHANNEL_BIX  
      resc_io: 75.00 resc_cpu: 552508  
      ix_sel: 1.000000 ix_sel_with_filters: 1.000000  
      Cost: 75.02 Resp: 75.02 Degree: 0  
  Access Path: index (FullScan)  
    Index: SALES_CUST_BIX  
      resc_io: 503.00 resc_cpu: 10743684
```

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

```
ix_sel: 1.000000 ix_sel_with_filters: 1.000000
Cost: 503.33 Resp: 503.33 Degree: 0
Access Path: index (FullScan)
Index: SALES_PROD_BIX
resc_io: 60.00 resc_cpu: 642086
ix_sel: 1.000000 ix_sel_with_filters: 1.000000
Cost: 60.02 Resp: 60.02 Degree: 0
Access Path: index (FullScan)
Index: SALES_PROMO_BIX
resc_io: 58.00 resc_cpu: 423844
ix_sel: 1.000000 ix_sel_with_filters: 1.000000
Cost: 58.01 Resp: 58.01 Degree: 0
Access Path: index (FullScan)
Index: SALES_TIME_BIX
resc_io: 60.00 resc_cpu: 719286
ix_sel: 1.000000 ix_sel_with_filters: 1.000000
Cost: 60.02 Resp: 60.02 Degree: 0
Access Path: index (FullScan)
Index: SALES_PROMO_BIX
resc_io: 58.00 resc_cpu: 423844
ix_sel: 1.000000 ix_sel_with_filters: 1.000000
Cost: 58.01 Resp: 58.01 Degree: 0
Bitmap nodes:
Used SALES_PROMO_BIX
Cost = 58.013101, sel = 1.000000
Access path: Bitmap index - accepted
Cost: 2881.534284 Cost_io: 2869.400000 Cost_cpu:
392576474.936000 Sel: 1.000000
Not Believed to be index-only
***** finished trying bitmap/domain indexes *****
Best:: AccessPath: TableScan
Cost: 489.06 Degree: 1 Resp: 489.06 Card: 918843.00
Bytes: 0

Access path analysis for TIMES
*****
SINGLE TABLE ACCESS PATH
Single Table Cardinality Estimation for TIMES [T]
Table: TIMES Alias: T
Card: Original: 1826.000000 Rounded: 183 Computed: 182.60
Non Adjusted: 182.60
Access Path: TableScan
Cost: 18.07 Resp: 18.07 Degree: 0
```

```
Cost_io: 18.00  Cost_cpu: 2314640
Resp_io: 18.00  Resp_cpu: 2314640
***** trying bitmap/domain indexes *****
Access Path: index (FullScan)
Index: TIMES_PK
resc_io: 6.00  resc_cpu: 407929
ix_sel: 1.000000  ix_sel_with_filters: 1.000000
Cost: 6.01  Resp: 6.01  Degree: 0
***** finished trying bitmap/domain indexes *****
Best:: AccessPath: TableScan
Cost: 18.07  Degree: 1  Resp: 18.07  Card: 182.60
Bytes: 0

Access path analysis for CUSTOMERS
*****
SINGLE TABLE ACCESS PATH
Single Table Cardinality Estimation for CUSTOMERS [C]
Table: CUSTOMERS  Alias: C
Card: Original: 55500.000000  Rounded: 383  Computed: 382.76
Non Adjusted: 382.76
Access Path: TableScan
Cost: 405.01  Resp: 405.01  Degree: 0
Cost_io: 404.00  Cost_cpu: 32782460
Resp_io: 404.00  Resp_cpu: 32782460
***** trying bitmap/domain indexes *****
Access Path: index (FullScan)
Index: CUSTOMERS_GENDER_BIX
resc_io: 4.00  resc_cpu: 29486
ix_sel: 1.000000  ix_sel_with_filters: 1.000000
Cost: 4.00  Resp: 4.00  Degree: 0
Access Path: index (FullScan)
Index: CUSTOMERS_MARITAL_BIX
resc_io: 6.00  resc_cpu: 46329
ix_sel: 1.000000  ix_sel_with_filters: 1.000000
Cost: 6.00  Resp: 6.00  Degree: 0
Access Path: index (FullScan)
Index: CUSTOMERS_PK
resc_io: 116.00  resc_cpu: 11926087
ix_sel: 1.000000  ix_sel_with_filters: 1.000000
Cost: 116.37  Resp: 116.37  Degree: 0
Access Path: index (FullScan)
Index: CUSTOMERS_YOB_BIX
resc_io: 20.00  resc_cpu: 157429
```

```
ix_sel: 1.000000 ix_sel_with_filters: 1.000000
Cost: 20.00 Resp: 20.00 Degree: 0
Access Path: index (FullScan)
Index: CUST_CUST_CITY_IDX
resc_io: 162.00 resc_cpu: 12253673
ix_sel: 1.000000 ix_sel_with_filters: 1.000000
Cost: 162.38 Resp: 162.38 Degree: 0
Access Path: index (FullScan)
Index: CUSTOMERS_GENDER_BIX
resc_io: 4.00 resc_cpu: 29486
ix_sel: 1.000000 ix_sel_with_filters: 1.000000
Cost: 4.00 Resp: 4.00 Degree: 0
Bitmap nodes:
Used CUSTOMERS_GENDER_BIX
Cost = 4.000911, sel = 1.000000
Access path: Bitmap index - accepted
Cost: 2365.912518 Cost_io: 2364.560000 Cost_cpu:
43757572.166400 Sel: 1.000000
Not Believed to be index-only
***** finished trying bitmap/domain indexes *****
Best:: AccessPath: TableScan
Cost: 405.01 Degree: 1 Resp: 405.01 Card: 382.76
Bytes: 0

Access path analysis for CHANNELS
*****
SINGLE TABLE ACCESS PATH
Single Table Cardinality Estimation for CHANNELS [CH]
Table: CHANNELS Alias: CH
Card: Original: 5.000000 Rounded: 2 Computed: 2.00 Non
Adjusted: 2.00
Access Path: TableScan
Cost: 3.00 Resp: 3.00 Degree: 0
Cost_io: 3.00 Cost_cpu: 29826
Resp_io: 3.00 Resp_cpu: 29826
***** trying bitmap/domain indexes *****
Access Path: index (FullScan)
Index: CHANNELS_PK
resc_io: 1.00 resc_cpu: 8121
ix_sel: 1.000000 ix_sel_with_filters: 1.000000
Cost: 1.00 Resp: 1.00 Degree: 0
***** finished trying bitmap/domain indexes *****
Best:: AccessPath: TableScan
```

```
Cost: 3.00  Degree: 1  Resp: 3.00  Card: 2.00  Bytes: 0

Grouping column cardinality [CHANNEL_CL]      2
Grouping column cardinality [ CUST_CITY]     286
Grouping column cardinality [CALENDAR_Q]      2
*****



OPTIMIZER STATISTICS AND COMPUTATIONS
*****
GENERAL PLANS
*****
Considering cardinality-based initial join order.
Permutations for Starting Table :0
Join order[1]: CHANNELS [CH] #0  TIMES [T] #1  CUSTOMERS [C] #2
SALES [S] #3

*****
Now joining: TIMES [T] #1
*****
NL Join
Outer table: Card: 2.00  Cost: 3.00  Resp: 3.00  Degree: 1
Bytes: 21
Access path analysis for TIMES
Inner table: TIMES  Alias: T
Access Path: TableScan
    NL Join: Cost: 37.14  Resp: 37.14  Degree: 1
        Cost_io: 37.00  Cost_cpu: 4659106
        Resp_io: 37.00  Resp_cpu: 4659106
    ***** trying bitmap/domain indexes *****
    Access Path: index (FullScan)
        Index: TIMES_PK
        resc_io: 6.00  resc_cpu: 407929
        ix_sel: 1.000000  ix_sel_with_filters: 1.000000
        Cost: 6.01  Resp: 6.01  Degree: 0
    ***** finished trying bitmap/domain indexes *****

    Best NL cost: 37.14
        resc: 37.14  resc_io: 37.00  resc_cpu: 4659106
        resp: 37.14  resp_io: 37.00  resp_cpu: 4659106
Join Card: 365.200000 = = outer (2.000000) * inner (182.600000)
* sel (1.000000)
Join Card - Rounded: 365 Computed: 365.20
```

```
Grouping column cardinality [CHANNEL_CL]          2
Grouping column cardinality [ CUST_CITY]         286
Grouping column cardinality [CALENDAR_Q]          2
Best::: JoinMethod: NestedLoop
      Cost: 37.14  Degree: 1  Resp: 37.14  Card: 365.20 Bytes:
37

*****
Now joining: CUSTOMERS[C] #2
*****
NL Join
  Outer table: Card: 365.20  Cost: 37.14  Resp: 37.14  Degree: 1
Bytes: 37
Access path analysis for CUSTOMERS
  Inner table: CUSTOMERS  Alias: C
  Access Path: TableScan
    NL Join:  Cost: 147305.99  Resp: 147305.99  Degree: 1
      Cost_io: 146936.00  Cost_cpu: 11970256947
      Resp_io: 146936.00  Resp_cpu: 11970256947
***** trying bitmap/domain indexes *****
  Access Path: index (FullScan)
    Index: CUSTOMERS_GENDER_BIX
    resc_io: 4.00  resc_cpu: 29486
    ix_sel: 1.000000  ix_sel_with_filters: 1.000000
    Cost: 4.00  Resp: 4.00  Degree: 0
  Access Path: index (FullScan)
    Index: CUSTOMERS_MARITAL_BIX
    resc_io: 6.00  resc_cpu: 46329
    ix_sel: 1.000000  ix_sel_with_filters: 1.000000
    Cost: 6.00  Resp: 6.00  Degree: 0
  Access Path: index (FullScan)
    Index: CUSTOMERS_PK
    resc_io: 116.00  resc_cpu: 11926087
    ix_sel: 1.000000  ix_sel_with_filters: 1.000000
    Cost: 116.37  Resp: 116.37  Degree: 0
  Access Path: index (FullScan)
    Index: CUSTOMERS_YOB_BIX
    resc_io: 20.00  resc_cpu: 157429
    ix_sel: 1.000000  ix_sel_with_filters: 1.000000
    Cost: 20.00  Resp: 20.00  Degree: 0
  Access Path: index (FullScan)
    Index: CUST_CUST_CITY_IDX
    resc_io: 162.00  resc_cpu: 12253673
```

```
ix_sel: 1.000000 ix_sel_with_filters: 1.000000
Cost: 162.38 Resp: 162.38 Degree: 0
Access Path: index (FullScan)
Index: CUSTOMERS_GENDER_BIX
resc_io: 4.00 resc_cpu: 29486
ix_sel: 1.000000 ix_sel_with_filters: 1.000000
NL Join : Cost: 1497.48 Resp: 1497.48 Degree: 1
Cost_io: 1497.00 Cost_cpu: 15421408
Resp_io: 1497.00 Resp_cpu: 15421408
Bitmap nodes:
Used CUSTOMERS_GENDER_BIX
Cost = 1497.476666, sel = 1.000000
Access path: Bitmap index - accepted
Cost: 863632.357158 Cost_io: 863138.400000 Cost_cpu:
15980832052.096001 Sel: 1.000000
Not Believed to be index-only
***** finished trying bitmap/domain indexes *****

Best NL cost: 147305.99
resc: 147305.99 resc_io: 146936.00 resc_cpu:
11970256947
resp: 147305.99 resp_io: 146936.00 resc_cpu:
11970256947
Join Card: 139783.448276 = = outer (365.200000) * inner
(382.758621) * sel (1.000000)
Join Card - Rounded: 139783 Computed: 139783.45
Grouping column cardinality [CHANNEL_CL] 2
Grouping column cardinality [CUST_CITY] 286
Grouping column cardinality [CALENDAR_Q] 2
Best:: JoinMethod: NestedLoop
Cost: 147305.99 Degree: 1 Resp: 147305.99 Card:
139783.45 Bytes: 63

*****
Now joining: SALES [S] #3
*****
NL Join
Outer table: Card: 139783.45 Cost: 147305.99 Resp: 147305.99
Degree: 1 Bytes: 63
Access path analysis for SALES
Inner table: SALES Alias: S
Access Path: TableScan
NL Join: Cost: 2579339.46 Resp: 2579339.46 Degree: 1
Cost_io: 2538743.82 Cost_cpu: 1313377131608
```

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

```
        Resp_io: 2538743.82  Resp_cpu: 1313377131608
***** trying bitmap/domain indexes *****
Access Path: index (AllEqJoinGuess)
Index: SALES_CHANNEL_BIX
resc_io: 11.00  resc_cpu: 83786
ix_sel: 0.250000  ix_sel_with_filters: 0.250000
NL Join : Cost: 1685281.00  Resp: 1685281.00  Degree: 1
    Cost_io: 1684549.00  Cost_cpu: 23682093020
    Resp_io: 1684549.00  Resp_cpu: 23682093020
Access Path: index (AllEqJoinGuess)
Index: SALES_CUST_BIX
resc_io: 1.00  resc_cpu: 9171
ix_sel: 0.000142  ix_sel_with_filters: 0.000142
NL Join : Cost: 287128.62  Resp: 287128.62  Degree: 1
    Cost_io: 286719.00  Cost_cpu: 13252268345
    Resp_io: 286719.00  Resp_cpu: 13252268345
Access Path: index (AllEqJoinGuess)
Index: SALES_TIME_BIX
resc_io: 1.00  resc_cpu: 8171
ix_sel: 0.000685  ix_sel_with_filters: 0.000685
NL Join : Cost: 287124.30  Resp: 287124.30  Degree: 1
    Cost_io: 286719.00  Cost_cpu: 13112485345
    Resp_io: 286719.00  Resp_cpu: 13112485345
Bitmap nodes:
Used SALES_TIME_BIX
Cost = 287124.298421, sel = 0.000685
Used SALES_CUST_BIX
Cost = 287128.619023, sel = 0.000142
Not used SALES_CHANNEL_BIX
Cost = 1685280.998142, sel = 0.250000
Access path: Bitmap index - accepted
Cost: 721678.844307 Cost_io: 720497.059076 Cost_cpu:
38233905490.990532 Sel: 0.000000
Not Believed to be index-only
***** finished trying bitmap/domain indexes *****

Best NL cost: 721678.84
    resc: 721678.84  resc_io: 720497.06  resc_cpu:
38233905491
    resp: 721678.84  resp_io: 720497.06  resp_cpu:
38233905491
Join Card: 3115.595241 = = outer (139783.448276) * inner
(918843.000000) * sel (0.000000)
```

```
Join Card - Rounded: 3116 Computed: 3115.60
Grouping column cardinality [CHANNEL_CL]      2
Grouping column cardinality [ CUST_CITY]      286
Grouping column cardinality [CALENDAR_Q]       2
Outer table: CUSTOMERS Alias: C
  resc: 147305.99   card 139783.45   bytes: 63   deg: 1   resp:
147305.99
Inner table: SALES Alias: S
  resc: 489.06    card: 918843.00   bytes: 21   deg: 1   resp:
489.06
  using dmeth: 2 #groups: 1
  SORT ressource          Sort statistics
    Sort width:           334 Area size:        292864 Max Area
size: 58720256
    Degree:                1
    Blocks to Sort: 1370 Row size:        80 Total Rows:
139783
    Initial runs: 2 Merge passes: 1 IO Cost / pass:
744
    Total IO sort cost: 2114           Total CPU sort cost:
173738577
    Total Temp space used: 21423000
  SORT ressource          Sort statistics
    Sort width:           334 Area size:        292864 Max Area
size: 58720256
    Degree:                1
    Blocks to Sort: 3825 Row size:        34 Total Rows:
918843
    Initial runs: 2 Merge passes: 1 IO Cost / pass:
2074
    Total IO sort cost: 5899           Total CPU sort cost:
946620547
    Total Temp space used: 66626000
  SM join: Resc: 155842.68  Resp: 155842.68
[multiMatchCost=0.00]
SM Join
  SM cost: 155842.68
    resc: 155842.68  resc_io: 155430.00  resc_cpu: 13351301509
    resp: 155842.68  resp_io: 155430.00  resp_cpu: 13351301509
Outer table: CUSTOMERS Alias: C
  resc: 147305.99   card 139783.45   bytes: 63   deg: 1   resp:
147305.99
Inner table: SALES Alias: S
  resc: 489.06    card: 918843.00   bytes: 21   deg: 1   resp:
489.06
```

```
using dmeth: 2 #groups: 1
Cost per ptn: 1936.46 #ptns: 1
hash_area: 124 (max=14336) buildfrag: 1280 probefrag: 3702
ppasses: 1
Hash join: Resc: 149731.51 Resp: 149731.51
[multiMatchCost=0.00]
HA Join
  HA cost: 149731.51
    resc: 149731.51 resc_io: 149346.00 resc_cpu: 12472293183
    resp: 149731.51 resp_io: 149346.00 resp_cpu: 12472293183
GROUP BY sort
GROUP BY adjustment factor: 0.500000
GROUP BY cardinality: 572.000000, TABLE cardinality:
3116.000000
  SORT ressource          Sort statistics
    Sort width:           334 Area size:       292864 Max Area
size:      58720256
    Degree:               1
    Blocks to Sort: 40 Row size:       103 Total Rows:
3116
    Initial runs:     1 Merge passes:  0 IO Cost / pass:
0
    Total IO sort cost: 0      Total CPU sort cost: 33981962
    Total Temp space used: 0
Best::: JoinMethod: Hash
  Cost: 149732.56 Degree: 1 Resp: 149732.56 Card:
3115.60 Bytes: 84
*****
Best so far: Table#: 0 cost: 3.0009 card: 2.0000 bytes: 42
              Table#: 1 cost: 37.1440 card: 365.2000 bytes:
13505
              Table#: 2 cost: 147305.9929 card: 139783.4483
bytes: 8806329
              Table#: 3 cost: 149732.5609 card: 3115.5952
bytes: 261744
*****
Join order[2]: CHANNELS [CH] #0 TIMES [T] #1 SALES [S] #3
CUSTOMERS [C] #2

*****
Now joining: SALES [S] #3
*****
NL Join
  Outer table: Card: 365.20 Cost: 37.14 Resp: 37.14 Degree: 1
Bytes: 37
```

```
Access path analysis for SALES
  Inner table: SALES Alias: S
  Access Path: TableScan
    NL Join: Cost: 6387.72 Resp: 6387.72 Degree: 1
      Cost_io: 6282.54 Cost_cpu: 3402879986
      Resp_io: 6282.54 Resp_cpu: 3402879986
***** trying bitmap/domain indexes *****
  Access Path: index (AllEqJoinGuess)
    Index: SALES_CHANNEL_BIX
    resc_io: 11.00 resc_cpu: 83786
    ix_sel: 0.250000 ix_sel_with_filters: 0.250000
    NL Join : Cost: 4053.09 Resp: 4053.09 Degree: 1
      Cost_io: 4052.00 Cost_cpu: 35240937
      Resp_io: 4052.00 Resp_cpu: 35240937
  Access Path: index (AllEqJoinGuess)
    Index: SALES_TIME_BIX
    resc_io: 1.00 resc_cpu: 8171
    ix_sel: 0.000685 ix_sel_with_filters: 0.000685
    NL Join : Cost: 402.24 Resp: 402.24 Degree: 1
      Cost_io: 402.00 Cost_cpu: 7641681
      Resp_io: 402.00 Resp_cpu: 7641681
  Bitmap nodes:
    Used SALES_TIME_BIX
      Cost = 402.236199, sel = 0.000685
    Used SALES_CHANNEL_BIX
      Cost = 4053.089275, sel = 0.250000
  Access path: Bitmap index - accepted
    Cost: 1390.849950 Cost_io: 1390.085842 Cost_cpu:
24720933.612843 Sel: 0.000171
    Not Believed to be index-only
***** finished trying bitmap/domain indexes *****

  Best NL cost: 1390.85
    resc: 1390.85 resc_io: 1390.09 resc_cpu: 24720934
    resp: 1390.85 resp_io: 1390.09 resp_cpu: 24720934
  Join Card: 57459.154726 = = outer (365.200000) * inner
(918843.000000) * sel (0.000171)
  Join Card - Rounded: 57459 Computed: 57459.15
  Grouping column cardinality [CHANNEL_CL] 2
  Grouping column cardinality [ CUST_CITY] 286
  Grouping column cardinality [CALENDAR_Q] 2
  Outer table: TIMES Alias: T
    resc: 37.14 card 365.20 bytes: 37 deg: 1 resp: 37.14
```

```
Inner table: SALES Alias: S
  resc: 489.06  card: 918843.00  bytes: 21  deg: 1  resp:
489.06
    using dmeth: 2 #groups: 1
    SORT ressource          Sort statistics
      Sort width:           334 Area size:       292864 Max Area
size: 58720256
      Degree:               1
      Blocks to Sort: 3 Row size:      51 Total Rows:
365
      Initial runs: 1 Merge passes: 0 IO Cost / pass:
0
      Total IO sort cost: 0      Total CPU sort cost: 32492643
      Total Temp space used: 0
    SORT ressource          Sort statistics
      Sort width:           334 Area size:       292864 Max Area
size: 58720256
      Degree:               1
      Blocks to Sort: 3825 Row size:      34 Total Rows:
918843
      Initial runs: 2 Merge passes: 1 IO Cost / pass:
2074
      Total IO sort cost: 5899      Total CPU sort cost:
946620547
      Total Temp space used: 66626000
    SM join: Resc: 6455.47  Resp: 6455.47  [multiMatchCost=0.00]
SM Join
    SM cost: 6455.47
      resc: 6455.47 resc_io: 6417.00 resc_cpu: 1244457733
      resp: 6455.47 resp_io: 6417.00 resp_cpu: 1244457733
Outer table: TIMES Alias: T
  resc: 37.14  card 365.20  bytes: 37  deg: 1  resp: 37.14
Inner table: SALES Alias: S
  resc: 489.06  card: 918843.00  bytes: 21  deg: 1  resp:
489.06
    using dmeth: 2 #groups: 1
    Cost per ptn: 3.34 #ptns: 1
    hash_area: 124 (max=14336) buildfrag: 3 probefrag: 3702
ppasses: 1
    Hash join: Resc: 529.54  Resp: 529.54  [multiMatchCost=0.00]
HA Join
    HA cost: 529.54
      resc: 529.54 resc_io: 518.00 resc_cpu: 373459927
      resp: 529.54 resp_io: 518.00 resp_cpu: 373459927
```

```
Best::: JoinMethod: Hash
      Cost: 529.54  Degree: 1  Resp: 529.54  Card: 57459.15
Bytes: 58

*****
Now joining: CUSTOMERS [C] #2
*****
NL Join
  Outer table: Card: 57459.15  Cost: 529.54  Resp: 529.54
Degree: 1  Bytes: 58
Access path analysis for CUSTOMERS
  Inner table: CUSTOMERS  Alias: C
  Access Path: TableScan
    NL Join:  Cost: 23183606.86  Resp: 23183606.86  Degree: 1
      Cost_io: 23125373.00  Cost_cpu: 1884020819874
      Resp_io: 23125373.00  Resp_cpu: 1884020819874
  Access Path: index (UniqueScan)
    Index: CUSTOMERS_PK
    resc_io: 1.00  resc_cpu: 9421
    ix_sel: 0.000018  ix_sel_with_filters: 0.000018
  NL Join : Cost: 58005.28  Resp: 58005.28  Degree: 1
    Cost_io: 57977.00  Cost_cpu: 914806448
    Resp_io: 57977.00  Resp_cpu: 914806448
  Access Path: index (AllEqUnique)
    Index: CUSTOMERS_PK
    resc_io: 1.00  resc_cpu: 9421
    ix_sel: 0.000018  ix_sel_with_filters: 0.000018
  NL Join : Cost: 58005.28  Resp: 58005.28  Degree: 1
    Cost_io: 57977.00  Cost_cpu: 914806448
    Resp_io: 57977.00  Resp_cpu: 914806448
***** trying bitmap/domain indexes *****
***** finished trying bitmap/domain indexes *****

Best NL cost: 58005.28
      resc: 58005.28  resc_io: 57977.00  resc_cpu: 914806448
      resp: 58005.28  resp_io: 57977.00  resp_cpu: 914806448
Join Card: 3115.595241 = = outer (57459.154726) * inner
(382.758621) * sel (0.000142)
Join Card - Rounded: 3116 Computed: 3115.60
Grouping column cardinality [CHANNEL_CL] 2
Grouping column cardinality [ CUST_CITY] 286
Grouping column cardinality [CALENDAR_Q] 2
Outer table: SALES  Alias: S
```

```
    resc: 529.54  card 57459.15  bytes: 58  deg: 1  resp: 529.54
Inner table: CUSTOMERS Alias: C
    resc: 405.01  card: 382.76  bytes: 26  deg: 1  resp: 405.01
    using dmeth: 2 #groups: 1
    SORT ressource          Sort statistics
        Sort width:      334 Area size:      292864 Max Area
size: 58720256
        Degree:           1
        Blocks to Sort: 521 Row size:       74 Total Rows:
57459
        Initial runs:   2 Merge passes:  1 IO Cost / pass:
284
        Total IO sort cost: 805      Total CPU sort cost: 86112225
        Total Temp space used: 8348000
    SORT ressource          Sort statistics
        Sort width:      334 Area size:      292864 Max Area
size: 58720256
        Degree:           1
        Blocks to Sort: 2 Row size:       39 Total Rows:
383
        Initial runs:   1 Merge passes:  0 IO Cost / pass:
0
        Total IO sort cost: 0      Total CPU sort cost: 32500745
        Total Temp space used: 0
    SM join: Resc: 1743.22  Resp: 1743.22  [multiMatchCost=0.00]
SM Join
    SM cost: 1743.22
        resc: 1743.22  resc_io: 1727.00  resc_cpu: 524855356
        resp: 1743.22  resp_io: 1727.00  resp_cpu: 524855356
Outer table: SALES Alias: S
    resc: 529.54  card 57459.15  bytes: 58  deg: 1  resp: 529.54
Inner table: CUSTOMERS Alias: C
    resc: 405.01  card: 382.76  bytes: 26  deg: 1  resp: 405.01
    using dmeth: 2 #groups: 1
    Cost per ptn: 192.83 #ptns: 1
    hash_area: 124 (max=14336) buildfrag: 491 probefrag: 2
ppasses: 1
    Hash join: Resc: 1127.40  Resp: 1127.40  [multiMatchCost=0.01]
Outer table: CUSTOMERS Alias: C
    resc: 405.01  card 382.76  bytes: 26  deg: 1  resp: 405.01
Inner table: SALES Alias: S
    resc: 529.54  card: 57459.15  bytes: 58  deg: 1  resp:
529.54
    using dmeth: 2 #groups: 1
```

```
Cost per ptn: 0.68 #ptns: 1
hash_area: 124 (max=14336) buildfrag: 2 probefrag: 491
ppasses: 1
Hash join: Resc: 935.24 Resp: 935.24 [multiMatchCost=0.00]
HA Join
  HA cost: 935.24 swapped
    resc: 935.24 resc_io: 922.00 resc_cpu: 428222071
    resp: 935.24 resp_io: 922.00 resp_cpu: 428222071
GROUP BY sort
GROUP BY adjustment factor: 0.500000
GROUP BY cardinality: 572.000000, TABLE cardinality:
3116.000000
  SORT ressource          Sort statistics
    Sort width:           334 Area size:        292864 Max Area
size: 58720256
    Degree:               1
    Blocks to Sort: 40 Row size:       103 Total Rows:
3116
    Initial runs: 1 Merge passes: 0 IO Cost / pass:
0
    Total IO sort cost: 0      Total CPU sort cost: 33981962
    Total Temp space used: 0
Best::: JoinMethod: Hash
  Cost: 936.29 Degree: 1 Resp: 936.29 Card: 3115.60
Bytes: 84
*****
Best so far: Table#: 0 cost: 3.0009 card: 2.0000 bytes: 42
  Table#: 1 cost: 37.1440 card: 365.2000 bytes:
13505
  Table#: 3 cost: 529.5434 card: 57459.1547
bytes: 3332622
  Table#: 2 cost: 936.2864 card: 3115.5952 bytes:
261744
*****
...
Join order[22]: SALES[S]#3 CUSTOMERS[C]#2 TIMES[T]#1
CHANNELS[CH]#0

*****
Now joining: TIMES[T]#1
*****
NL Join
```

```
Outer table: Card: 49822.22  Cost: 897.41  Resp: 897.41
Degree: 1  Bytes: 47
Access path analysis for TIMES
  Inner table: TIMES  Alias: T
  Access Path: TableScan
    NL Join:  Cost: 800577.88  Resp: 800577.88  Degree: 1
      Cost_io: 797001.00  Cost_cpu: 115721578068
      Resp_io: 797001.00  Resp_cpu: 115721578068
  Access Path: index (UniqueScan)
    Index: TIMES_PK
    resc_io: 1.00  resc_cpu: 10059
    ix_sel: 0.000548  ix_sel_with_filters: 0.000548
    NL Join : Cost: 50734.90  Resp: 50734.90  Degree: 1
      Cost_io: 50707.00  Cost_cpu: 902742490
      Resp_io: 50707.00  Resp_cpu: 902742490
  Access Path: index (AllEqUnique)
    Index: TIMES_PK
    resc_io: 1.00  resc_cpu: 10059
    ix_sel: 0.000548  ix_sel_with_filters: 0.000548
    NL Join : Cost: 50734.90  Resp: 50734.90  Degree: 1
      Cost_io: 50707.00  Cost_cpu: 902742490
      Resp_io: 50707.00  Resp_cpu: 902742490
***** trying bitmap/domain indexes *****
***** finished trying bitmap/domain indexes *****

Best NL cost: 50734.90
  resc: 50734.90  resc_io: 50707.00  resc_cpu: 902742490
  resp: 50734.90  resp_io: 50707.00  resp_cpu: 902742490
Join Card: 6231.190483 = = outer (49822.224013) * inner
(182.600000) * sel (0.000685)
Join Card - Rounded: 6231 Computed: 6231.19
Grouping column cardinality [CHANNEL_CL]      2
Grouping column cardinality [ CUST_CITY]     286
Grouping column cardinality [CALENDAR_Q]      2
  Outer table: CUSTOMERS  Alias: C
    resc: 897.41  card 49822.22  bytes: 47  deg: 1  resp: 897.41
  Inner table: TIMES  Alias: T
    resc: 18.07  card: 182.60  bytes: 16  deg: 1  resp: 18.07
    using dmeth: 2  #groups: 1
    SORT ressource          Sort statistics
      Sort width:           334 Area size:        292864 Max Area
size: 58720256
Degree: 1
```

```
Blocks to Sort: 379 Row size:       62 Total Rows:
49822
Initial runs:    2 Merge passes:  1 IO Cost / pass:
206
Total IO sort cost: 585      Total CPU sort cost: 76713467
Total Temp space used: 6022000
SORT ressource          Sort statistics
Sort width:           334 Area size:        292864 Max Area
size:    58720256
Degree:               1
Blocks to Sort: 1 Row size:       28 Total Rows:
183
Initial runs:    1 Merge passes:  0 IO Cost / pass:
0
Total IO sort cost: 0      Total CPU sort cost: 32414635
Total Temp space used: 0
SM join: Resc: 1503.86  Resp: 1503.86  [multiMatchCost=0.00]
SM Join
SM cost: 1503.86
resc: 1503.86 resc_io: 1488.00 resc_cpu: 513028724
resp: 1503.86 resp_io: 1488.00 resp_cpu: 513028724
Outer table: CUSTOMERS Alias: C
resc: 897.41 card 49822.22 bytes: 47 deg: 1 resp: 897.41
Inner table: TIMES Alias: T
resc: 18.07 card: 182.60 bytes: 16 deg: 1 resp: 18.07
using dmeth: 2 #groups: 1
Cost per ptn: 140.78 #ptns: 1
hash_area: 124 (max=14336) buildfrag: 359 probefrag: 1
ppasses: 1
Hash join: Resc: 1056.28  Resp: 1056.28  [multiMatchCost=0.02]
Outer table: TIMES Alias: T
resc: 18.07 card 182.60 bytes: 16 deg: 1 resp: 18.07
Inner table: CUSTOMERS Alias: C
resc: 897.41 card: 49822.22 bytes: 47 deg: 1 resp:
897.41
using dmeth: 2 #groups: 1
Cost per ptn: 0.65 #ptns: 1
hash_area: 124 (max=14336) buildfrag: 1 probefrag: 359
ppasses: 1
Hash join: Resc: 916.14  Resp: 916.14  [multiMatchCost=0.00]
HA Join
HA cost: 916.14 swapped
resc: 916.14 resc_io: 903.00 resc_cpu: 425086605
resp: 916.14 resp_io: 903.00 resp_cpu: 425086605
```

```
Best::: JoinMethod: Hash
      Cost: 916.14  Degree: 1  Resp: 916.14  Card: 6231.19
Bytes: 63

*****
Now joining: CHANNELS [CH] #0
*****
NL Join
  Outer table: Card: 6231.19  Cost: 916.14  Resp: 916.14
Degree: 1  Bytes: 63
Access path analysis for CHANNELS
  Inner table: CHANNELS  Alias: CH
  Access Path: TableScan
    NL Join:  Cost: 7673.88  Resp: 7673.88  Degree: 1
      Cost_io: 7655.00  Cost_cpu: 610930916
      Resp_io: 7655.00  Resp_cpu: 610930916
    Access Path: index (UniqueScan)
      Index: CHANNELS_PK
      resc_io: 1.00  resc_cpu: 8451
      ix_sel: 0.200000  ix_sel_with_filters: 0.200000
    NL Join : Cost: 7148.77  Resp: 7148.77  Degree: 1
      Cost_io: 7134.00  Cost_cpu: 477747528
      Resp_io: 7134.00  Resp_cpu: 477747528
    Access Path: index (AllEqUnique)
      Index: CHANNELS_PK
      resc_io: 1.00  resc_cpu: 8451
      ix_sel: 0.200000  ix_sel_with_filters: 0.200000
    NL Join : Cost: 7148.77  Resp: 7148.77  Degree: 1
      Cost_io: 7134.00  Cost_cpu: 477747528
      Resp_io: 7134.00  Resp_cpu: 477747528
***** trying bitmap/domain indexes *****
***** finished trying bitmap/domain indexes *****

Best NL cost: 7148.77
      resc: 7148.77  resc_io: 7134.00  resc_cpu: 477747528
      resp: 7148.77  resp_io: 7134.00  resp_cpu: 477747528
Join Card: 3115.595241 = = outer (6231.190483) * inner
(2.000000) * sel (0.250000)
Join Card - Rounded: 3116 Computed: 3115.60
Grouping column cardinality [CHANNEL_CL] 2
Grouping column cardinality [ CUST_CITY] 286
Grouping column cardinality [CALENDAR_Q] 2
Outer table: TIMES  Alias: T
```

```
    resc: 916.14  card 6231.19  bytes: 63  deg: 1  resp: 916.14
Inner table: CHANNELS Alias: CH
    resc: 3.00  card: 2.00  bytes: 21  deg: 1  resp: 3.00
    using dmeth: 2 #groups: 1
        SORT ressource          Sort statistics
            Sort width:      334 Area size:      292864 Max Area
size: 58720256
            Degree:           1
            Blocks to Sort: 62 Row size:       80 Total Rows:
6231
            Initial runs:   2 Merge passes:  1 IO Cost / pass:
36
            Total IO sort cost: 98      Total CPU sort cost: 37418215
            Total Temp space used: 926000
        SORT ressource          Sort statistics
            Sort width:      334 Area size:      292864 Max Area
size: 58720256
            Degree:           1
            Blocks to Sort: 1 Row size:       34 Total Rows:
2
            Initial runs:   1 Merge passes:  0 IO Cost / pass:
0
            Total IO sort cost: 0      Total CPU sort cost: 32352758
            Total Temp space used: 0
SM join: Resc: 1019.30  Resp: 1019.30  [multiMatchCost=0.00]
SM Join
    SM cost: 1019.30
        resc: 1019.30  resc_io: 1004.00  resc_cpu: 494887404
        resp: 1019.30  resp_io: 1004.00  resp_cpu: 494887404
Outer table: TIMES Alias: T
    resc: 916.14  card 6231.19  bytes: 63  deg: 1  resp: 916.14
Inner table: CHANNELS Alias: CH
    resc: 3.00  card: 2.00  bytes: 21  deg: 1  resp: 3.00
    using dmeth: 2 #groups: 1
    Cost per ptn: 0.53 #ptns: 1
    hash_area: 124 (max=14336) buildfrag: 58 probefrag: 1
ppasses: 1
    Hash join: Resc: 919.68  Resp: 919.68  [multiMatchCost=0.01]
Outer table: CHANNELS Alias: CH
    resc: 3.00  card 2.00  bytes: 21  deg: 1  resp: 3.00
Inner table: TIMES Alias: T
    resc: 916.14  card: 6231.19  bytes: 63  deg: 1  resp: 916.14
    using dmeth: 2 #groups: 1
    Cost per ptn: 0.52 #ptns: 1
```

```
hash_area: 124 (max=14336) buildfrag: 1 probefrag: 58
ppasses: 1
Hash join: Resc: 919.66 Resp: 919.66 [multiMatchCost=0.00]
HA Join
  HA cost: 919.66 swapped
    resc: 919.66 resc_io: 906.00 resc_cpu: 441916165
    resp: 919.66 resp_io: 906.00 resp_cpu: 441916165
GROUP BY sort
GROUP BY adjustment factor: 0.500000
GROUP BY cardinality: 572.000000, TABLE cardinality:
3116.000000
  SORT ressource          Sort statistics
    Sort width:           334 Area size:      292864 Max Area
size: 58720256
    Degree:               1
    Blocks to Sort: 40 Row size:      103 Total Rows:
3116
    Initial runs: 1 Merge passes: 0 IO Cost / pass:
0
    Total IO sort cost: 0      Total CPU sort cost: 33981962
    Total Temp space used: 0
Join order aborted: cost > best plan cost
*****
(newjo-stop-1) k:0, spcnt:0, perm:22, maxperm:2000

*****
Number of join permutations tried: 22
*****
Consider using bloom filter between C[CUSTOMERS] and S[SALES]
kkoBloomFilter: join (lcdn:383 rcdn:918843 jcdn:49822
limit:175847540)
Computing bloom ndv for creator:C[CUSTOMERS] ccdn:382.8 and
user:S[SALES] ucdn:918843.0
kkopqComputeBloomNdv: predicate (bndv:7059 ndv:7059) and
(bndv:55500 ndv:370)
kkopqComputeBloomNdv: pred cnt:2 ndv:383 reduction:0
kkoBloomFilter: join ndv:383 reduction:0.000417 (limit:0.500000)
accepted invalidated
Consider using bloom filter between S[SALES] and T[TIMES] ,with
join inputs swapped
kkoBloomFilter: join (lcdn:49822 rcdn:183 jcdn:6231
limit:4548769)
Computing bloom ndv for creator:T[TIMES] ccdn:182.6 and
user:S[SALES] ucdn:49822.2
```

```
kkopqComputeBloomNdv: predicate (bndv:1460 ndv:1460) and  
(bndv:1826 ndv:183)  
kkopqComputeBloomNdv: pred cnt:2 ndv:183 reduction:0  
kkoBloomFilter: join ndv:183 reduction:0.003665 (limit:0.500000)  
accepted invalidated  
Consider using bloom filter between T[TIMES] and CH[CHANNELS]  
, with join inputs swapped  
kkoBloomFilter: join (lcdn:6231 rcdn:2 jcdn:3116 limit:6231)  
Computing bloom ndv for creator:CH[CHANNELS] ccdn:2.0 and  
user:T[TIMES] ucdn:6231.2  
kkopqComputeBloomNdv: predicate (bndv:4 ndv:4) and (bndv:5  
ndv:2)  
kkopqComputeBloomNdv: pred cnt:2 ndv:2 reduction:0  
kkoBloomFilter: join ndv:2 reduction:0.000321 (limit:0.500000)  
accepted invalidated  
(newjo-save) [1 3 2 0 ]  
GROUP BY adjustment factor: 0.500000  
GROUP BY cardinality: 572.000000, TABLE cardinality:  
3116.000000  
    SORT ressource                Sort statistics  
    Sort width:                334 Area size:          292864 Max Area  
size:      58720256  
    Degree:                      1  
    Blocks to Sort: 40 Row size:      103 Total Rows:  
3116  
    Initial runs:      1 Merge passes:  0 IO Cost / pass:  
0  
    Total IO sort cost: 0      Total CPU sort cost: 33981962  
    Total Temp space used: 0  
Trying or-Expansion on query block SEL$1 (#1)  
Transfer Optimizer annotations for query block SEL$1 (#1)  
id=0 frofand predicate="C"."CUST_STATE_PROVINCE"='CA'  
id=0 frofksm[i] (sort-merge/hash)  
predicate="S"."CUST_ID"="C"."CUST_ID"  
id=0 frosand (sort-merge/hash)  
predicate="S"."CUST_ID"="C"."CUST_ID"  
id=0 frofksm[i] (sort-merge/hash)  
predicate="S"."TIME_ID"="T"."TIME_ID"  
id=0 frosand (sort-merge/hash)  
predicate="S"."TIME_ID"="T"."TIME_ID"  
id=0 frofand predicate="T"."CALENDAR_QUARTER_DESC"='1999-Q1' OR  
"T"."CALENDAR_QUARTER_DESC"='1999-Q2'  
id=0 frofksm[i] (sort-merge/hash)  
predicate="S"."CHANNEL_ID"="CH"."CHANNEL_ID"  
id=0 frosand (sort-merge/hash)  
predicate="S"."CHANNEL_ID"="CH"."CHANNEL_ID"
```

```
id=0 frofand predicate="CH"."CHANNEL_DESC"='Catalog' OR
"CH"."CHANNEL_DESC"='Internet'
GROUP BY adjustment factor: 1.000000
Final cost for query block SEL$1 (#1) - All Rows Plan:
  Best join order: 16
  Cost: 920.7097  Degree: 1  Card: 3116.0000  Bytes: 261744
  Resc: 920.7097  Resc_io: 906.0000  Resc_cpu: 475898127
  Resp: 920.7097  Resp_io: 906.0000  Resc_cpu: 475898127
  kkoqbc-subheap (delete addr=0x61b144, in-use=119500,
alloc=135000)
  kkoqbc-end:
    :
    call(in-use=131436, alloc=284540), compile(in-use=147780,
alloc=150188), execution(in-use=3504, alloc=4060)

  kkoqbc: finish optimizing query block SEL$1 (#1)
  apadrv-end
    :
    call(in-use=131436, alloc=284540), compile(in-use=148496,
alloc=150188), execution(in-use=3504, alloc=4060)

Starting SQL statement dump

user_id=0 user_name=SYS module=sqlplus@EDRSR10P1 (TNS V1-V3)
action=
sql_id=70fqjd9u1zk7c plan_hash_value=593420798 problem_type=3
----- Current SQL Statement for this session
(sql_id=70fqjd9u1zk7c) -----
SELECT ch.channel_class, c.cust_city, t.calendar_quarter_desc,
SUM(s.amount_sold) sales_amount
FROM sh.sales s,sh.times t,sh.customers c,sh.channels ch
WHERE s.time_id = t.time_id AND
s.cust_id = c.cust_id AND
s.channel_id = ch.channel_id AND
c.cust_state_province = 'CA' AND
ch.channel_desc IN ('Internet','Catalog') AND
t.calendar_quarter_desc IN ('1999-Q1','1999-Q2')
GROUP BY ch.channel_class, c.cust_city, t.calendar_quarter_desc
sql_text_length=473
sql=SELECT ch.channel_class, c.cust_city,
t.calendar_quarter_desc, SUM(s.amount_sold) sales_amount
FROM sh.sales s,sh.times t,sh.customers c,sh.channels ch
WHERE s.time_id = t.time_id AND
```

```

        s.cust_id = c.cust_id AND
        s.channel_id = ch.channel_id
sql=AND
        c.cust_state_province = 'CA' AND
        ch.channel_desc IN ('Internet', 'Catalog') AND
        t.calendar_quarter_desc IN ('1999-Q1', '1999-Q2')
GROUP BY ch.channel_class, c.cust_city, t.calendar_quarter_desc
----- Explain Plan Dump -----
----- Plan Table -----

=====
Plan Table
=====

-----+-----+
| Id | Operation           | Name | Rows |
Bytes | Cost   | Time      | Pstart| Pstop |
-----+-----+
-----+-----+
| 0  | SELECT STATEMENT    |       |       |
| 921 |             |       |       |       |
| 1  | HASH GROUP BY       |       |       | 572  |
47K  | 921  | 00:00:12 |       |       |
| 2  | HASH JOIN            |       |       | 3116 |
256K | 920  | 00:00:12 |       |       |
| 3  | TABLE ACCESS FULL   | CHANNELS | 2  |
42   | 3   | 00:00:01 |       |       |
| 4  | HASH JOIN            |       |       | 6231 |
383K | 916  | 00:00:11 |       |       |
| 5  | PART JOIN FILTER CREATE | :BF0000 | 183 |
2928 | 18   | 00:00:01 |       |       |
| 6  | TABLE ACCESS FULL   | TIMES | 183 |
2928 | 18   | 00:00:01 |       |       |
| 7  | HASH JOIN            |       |       | 49K  |
2287K | 897  | 00:00:11 |       |       |
| 8  | TABLE ACCESS FULL   | CUSTOMERS | 383 |
9958 | 405  | 00:00:05 |       |       |
| 9  | PARTITION RANGE JOIN-FILTER |       | 897K |
18M  | 489  | 00:00:06 | :BF0000| :BF0000|
| 10 | TABLE ACCESS FULL   | SALES | 897K |
18M  | 489  | 00:00:06 | :BF0000| :BF0000|
-----+-----+
-----+-----+
Predicate Information:
-----
```

```

2 - access ("S"."CHANNEL_ID"="CH"."CHANNEL_ID")
3 - filter(("CH"."CHANNEL_DESC"='Catalog' OR
"CH"."CHANNEL_DESC"='Internet'))
4 - access ("S"."TIME_ID"="T"."TIME_ID")
6 - filter(("T"."CALENDAR_QUARTER_DESC"='1999-Q1' OR
"T"."CALENDAR_QUARTER_DESC"='1999-Q2'))
7 - access ("S"."CUST_ID"="C"."CUST_ID")
8 - filter("C"."CUST_STATE_PROVINCE"='CA')

Content of other_xml column
=====
nodeid/pflags: 10 513nodeid/pflags: 9 513 db_version      :
11.2.0.1
parse_schema      : SYS
plan_hash         : 593420798
plan_hash_2       : 1128146253
Outline Data:
/*
  BEGIN_OUTLINE_DATA
    IGNORE_OPTIM_EMBEDDED_HINTS
    OPTIMIZER_FEATURES_ENABLE('12.1.0.1')
    DB_VERSION('12.1.0.1')
    ALL_ROWS
    OUTLINE_LEAF(@"SEL$1")
    FULL(@"SEL$1" "C@"SEL$1")
    FULL(@"SEL$1" "S@"SEL$1")
    FULL(@"SEL$1" "T@"SEL$1")
    FULL(@"SEL$1" "CH@"SEL$1")
    LEADING(@"SEL$1" "C@"SEL$1" "S@"SEL$1" "T@"SEL$1"
"CH@"SEL$1")
    USE_HASH(@"SEL$1" "S@"SEL$1")
    USE_HASH(@"SEL$1" "T@"SEL$1")
    USE_HASH(@"SEL$1" "CH@"SEL$1")
    PX_JOIN_FILTER(@"SEL$1" "T@"SEL$1")
    SWAP_JOIN_INPUTS(@"SEL$1" "T@"SEL$1")
    SWAP_JOIN_INPUTS(@"SEL$1" "CH@"SEL$1")
    USE_HASH_AGGREGATION(@"SEL$1")
  END_OUTLINE_DATA
*/
Optimizer state dump:
Compilation Environment Dump
optimizer_mode_hinted          = false

```

```
optimizer_features_hinted          = 0.0.0
parallel_execution_enabled         = true
parallel_query_forced_dop          = 0
parallel_dml_forced_dop           = 0
parallel_ddl_forced_degree        = 0
parallel_ddl_forced_instances     = 0
_query_rewrite_fudge              = 90
optimizer_features_enable          = 12.1.0.1
...
Bug Fix Control Environment
fix 3834770 = 1
fix 3746511 = enabled
fix 4519016 = enabled
fix 3118776 = enabled
fix 4488689 = enabled
fix 2194204 = disabled
...
Query Block Registry:
SEL$1 0x5c3a04 (PARSER) [FINAL]

:
call(in-use=150804, alloc=284540), compile(in-use=182764,
alloc=243324), execution(in-use=15240, alloc=16288)

End of Optimizer State Dump
Dumping Hints
=====
===== END SQL Statement Dump
=====

$
```

3. Execute the `te_cleanup.sh` script to clean up your environment for this practice.

```
$ cd /home/oracle/labs/solutions/Trace_Event
$ ./te_cleanup.sh
```

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Error : You are not a Valid Partner use only

Practices for Lesson 6: Generating and Displaying Execution Plans

Chapter 6

Practices for Lesson 6: Overview

Practices Overview

In these practices, you use SQL Developer to create and view execution plans and use SQL*Plus to create and retrieve view execution plans from various sources.

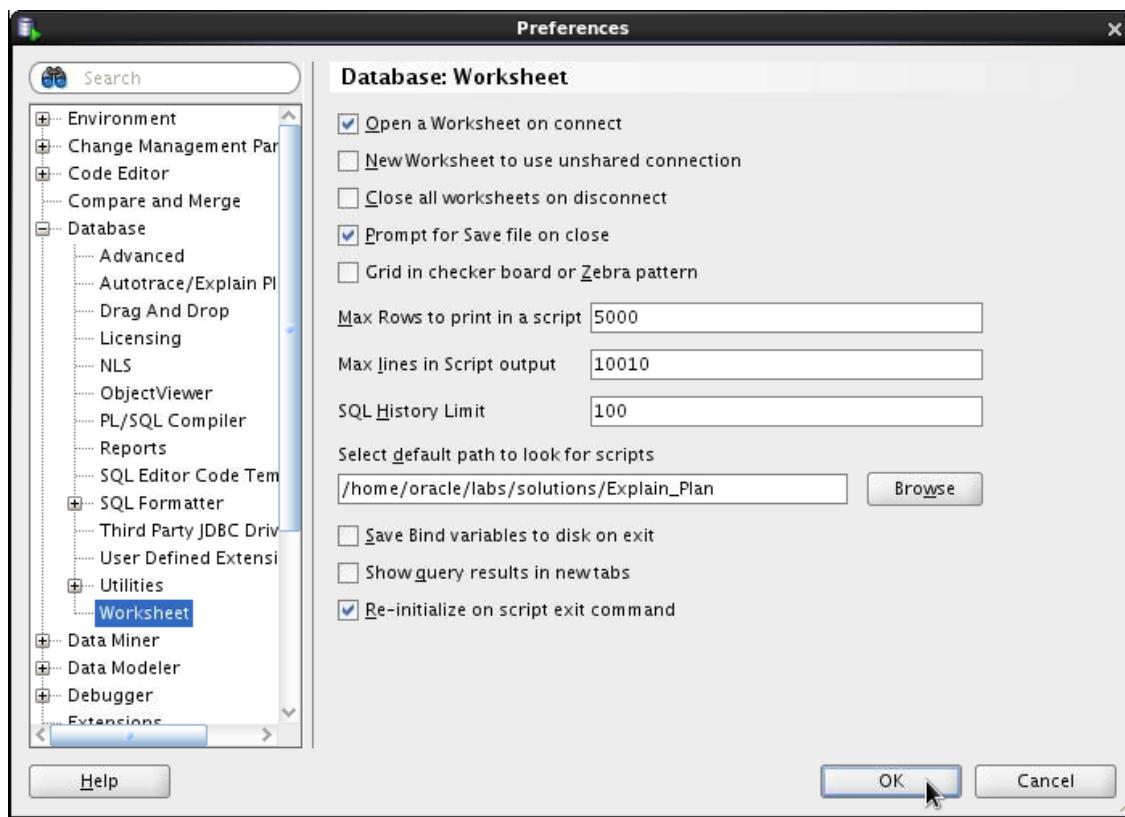
Practice 6-1: Extracting an Execution Plan by Using SQL Developer

Overview

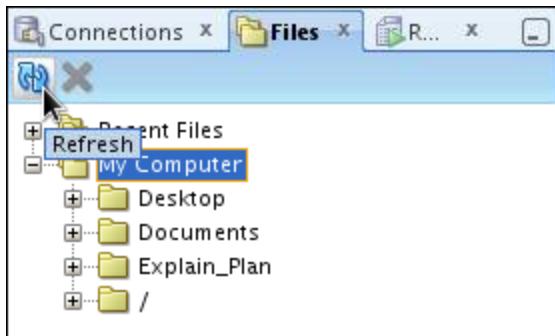
In this practice, you use SQL Developer to display the execution plan of a SQL query, and then use the Autotrace option.

Tasks

1. The scripts for this practice are in the `/home/oracle/labs/solutions/Explain_Plan` directory. Set the SQL worksheet preferences to set the default directory.
 - a. From the menu, select Tools > Preferences, expand the Database item, and select Worksheet. On the Worksheet, browse for the `/home/oracle/labs/solutions/Explain_Plan` directory. Click Select, and then click OK.



- b. In the navigator pane, click the Files tab. Select My Computer and click the Refresh button. The default Worksheet directory appears in the list.



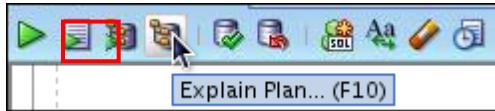
2. Display the explain plan for the following query using scott user:

- a. On the SQL worksheet, enter the following command:

```
SELECT ename, job, sal, dname  
FROM emp, dept  
WHERE dept.deptno = emp.deptno;
```

You may instead open the ep_join.sql script in the /home/oracle/labs/solutions/Explain_Plan directory.

- b. Click the Explain Plan button.



- c. Note the explain plan.

The screenshot shows the Oracle SQL Developer interface. In the top navigation bar, there are tabs for 'Start Page', 'ep_join.sql' (which is selected), and 'ep_not_exists.sql'. Below the tabs is a toolbar with various icons for database operations like 'Run', 'Stop', 'Autotrace', and 'Explain Plan'. The main workspace is divided into two panes. The left pane contains the SQL worksheet with the following query:

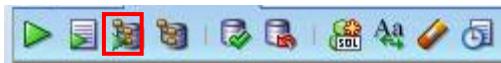
```
SELECT ename, job, sal, dname
FROM emp, dept
WHERE dept.deptno = emp.deptno;
```

The right pane displays the execution plan for this query. The plan is a tree structure starting with a 'SELECT STATEMENT' node, which branches into a 'MERGE JOIN' node. This join node has two children: a 'TABLE ACCESS' node for the 'DEPT' table (using an 'INDEX' scan) and a 'SORT' node. The 'SORT' node has two children: 'Access Predicates' (showing the condition 'DEPT.DEPTNO=EMP.DEPTNO') and 'Filter Predicates' (also showing the same condition). Finally, the 'SORT' node connects to another 'TABLE ACCESS' node for the 'EMP' table (using a 'FULL' scan). The 'OPTIONS' column for the 'DEPT' access path indicates 'BY INDEX ROWID' and 'FULL SCAN'. The 'OPTIONS' column for the 'EMP' access path indicates 'JOIN'.

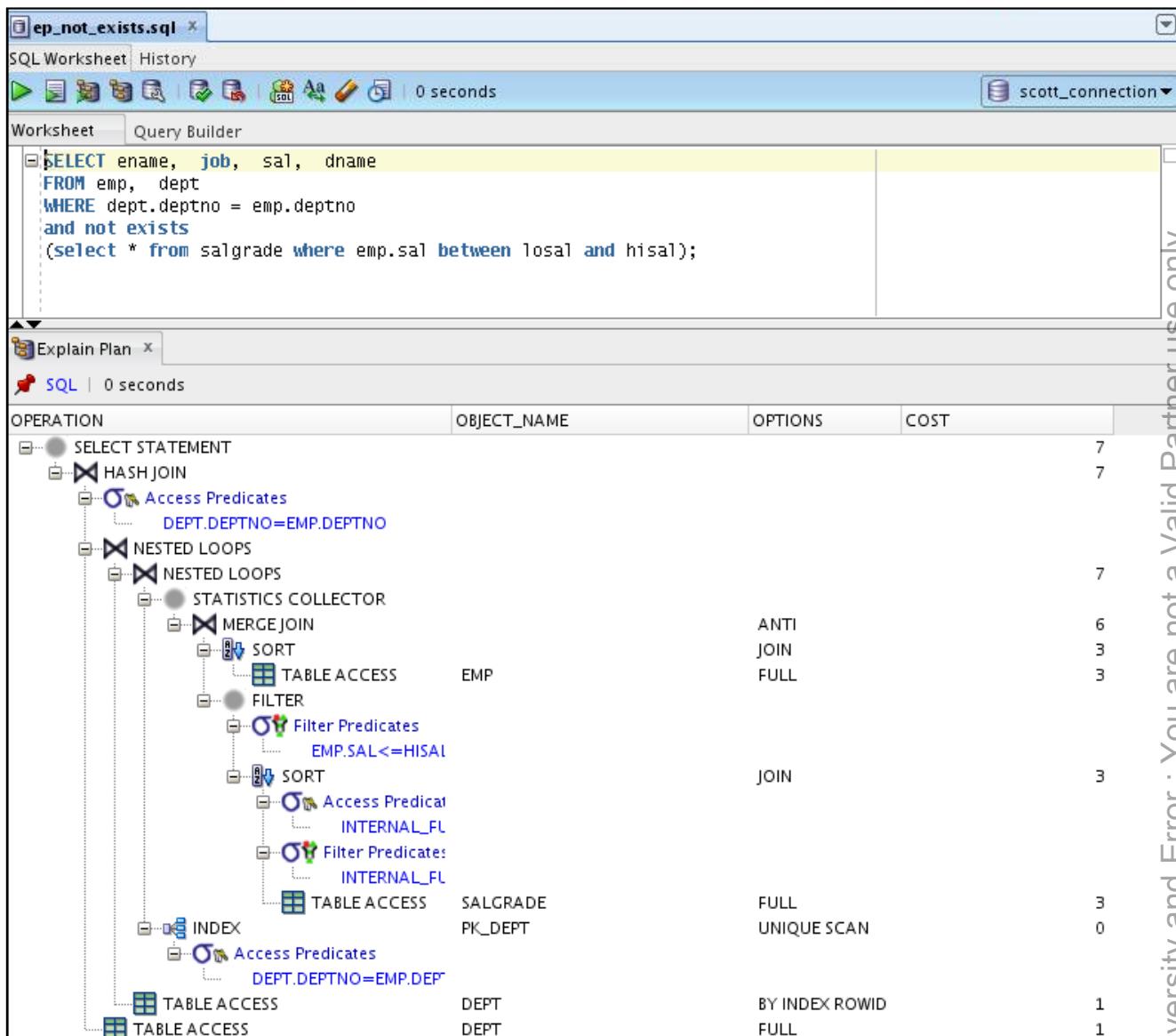
3. Change the query to the following and note the difference in the explain plan. You may instead open the `ep_not_exists.sql` script.

```
SELECT ename, job, sal, dname
FROM emp, dept
WHERE dept.deptno = emp.deptno
and not exists
(select * from salgrade where emp.sal between losal and hisal);
```

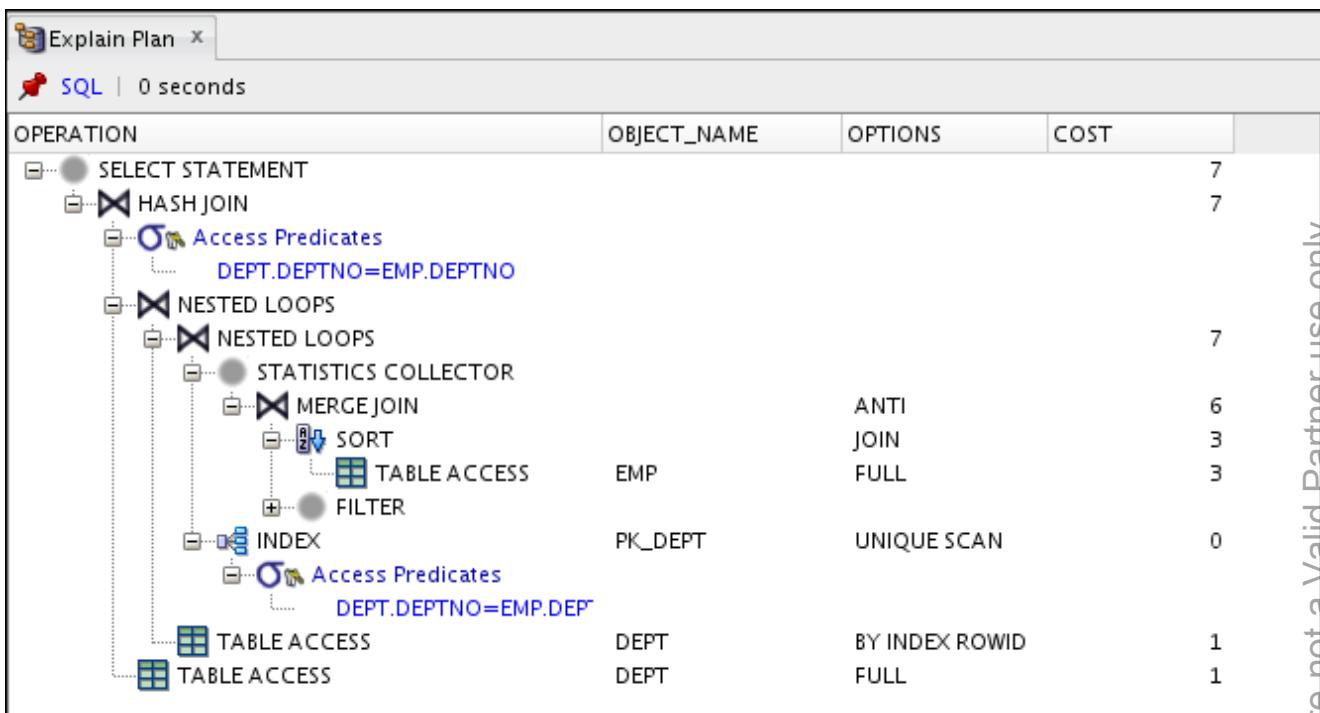
- On the SQL worksheet, enter the preceding query.
- Click the Explain Plan button.



- c. Note the difference in the explain plan.

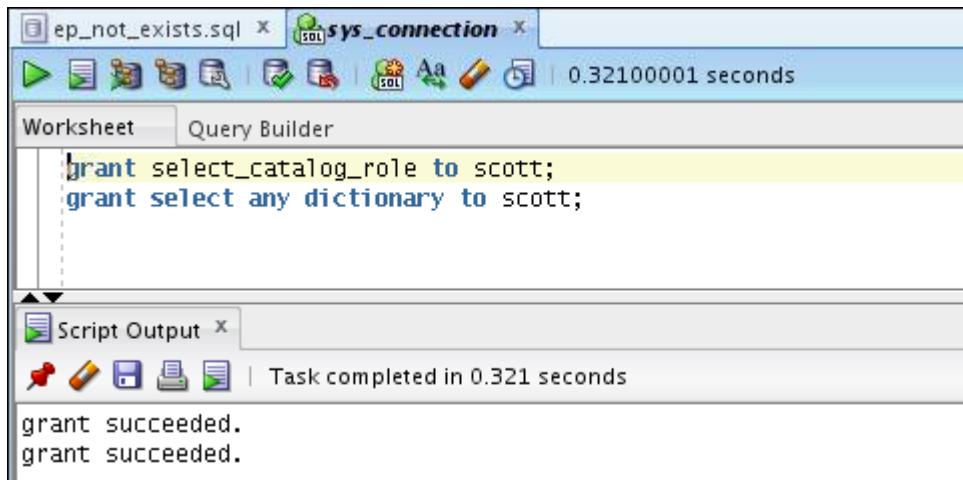


- d. Collapse the information about the filters.



4. From sys_connection, grant select_catalog_role and Select Any Dictionary to scott.

```
grant select_catalog_role to scott;
grant select any dictionary to scott;
```

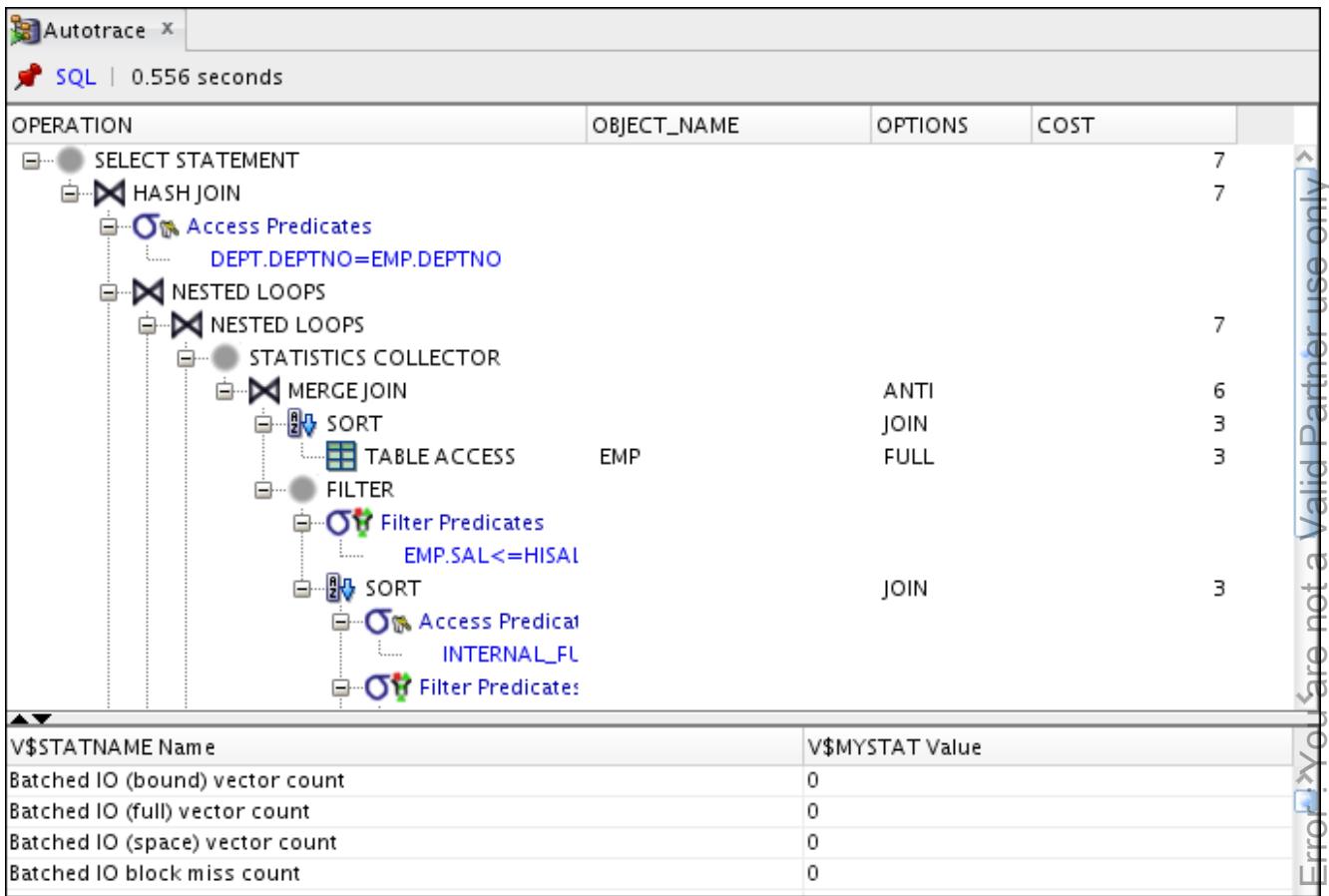


5. Display Autotrace for the query in the ep_not_exists.sql script.

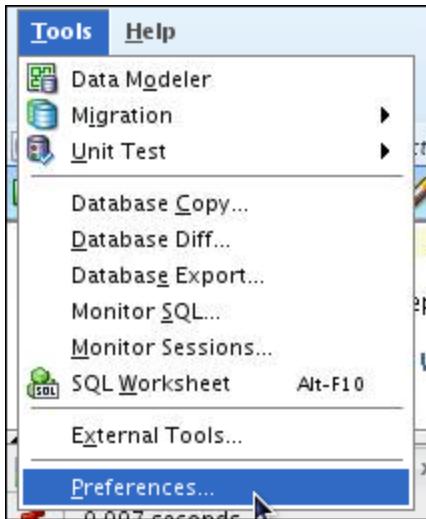
- a. In the scott connection, click the Autotrace button.



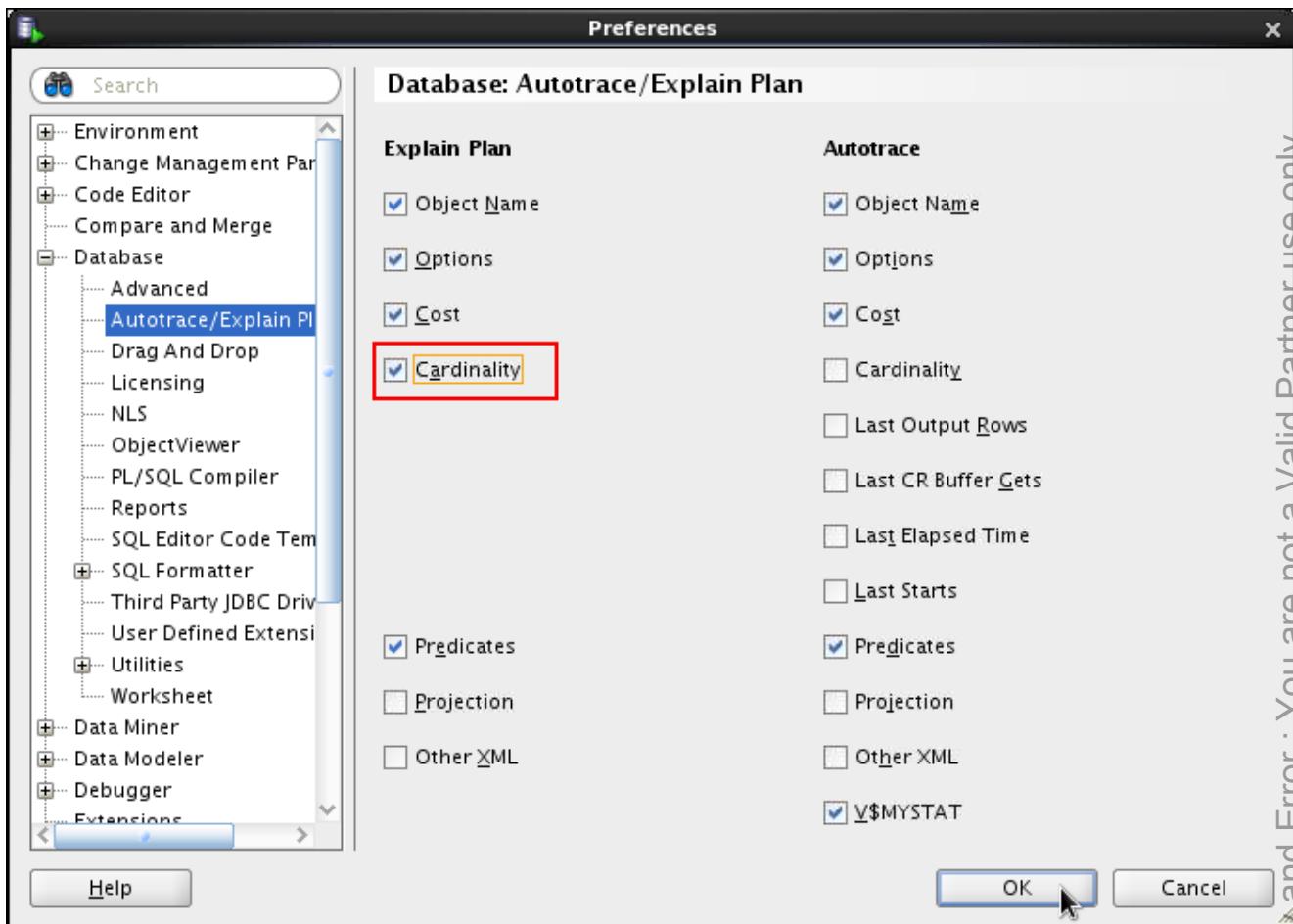
- b. Note the Autotrace output tab.



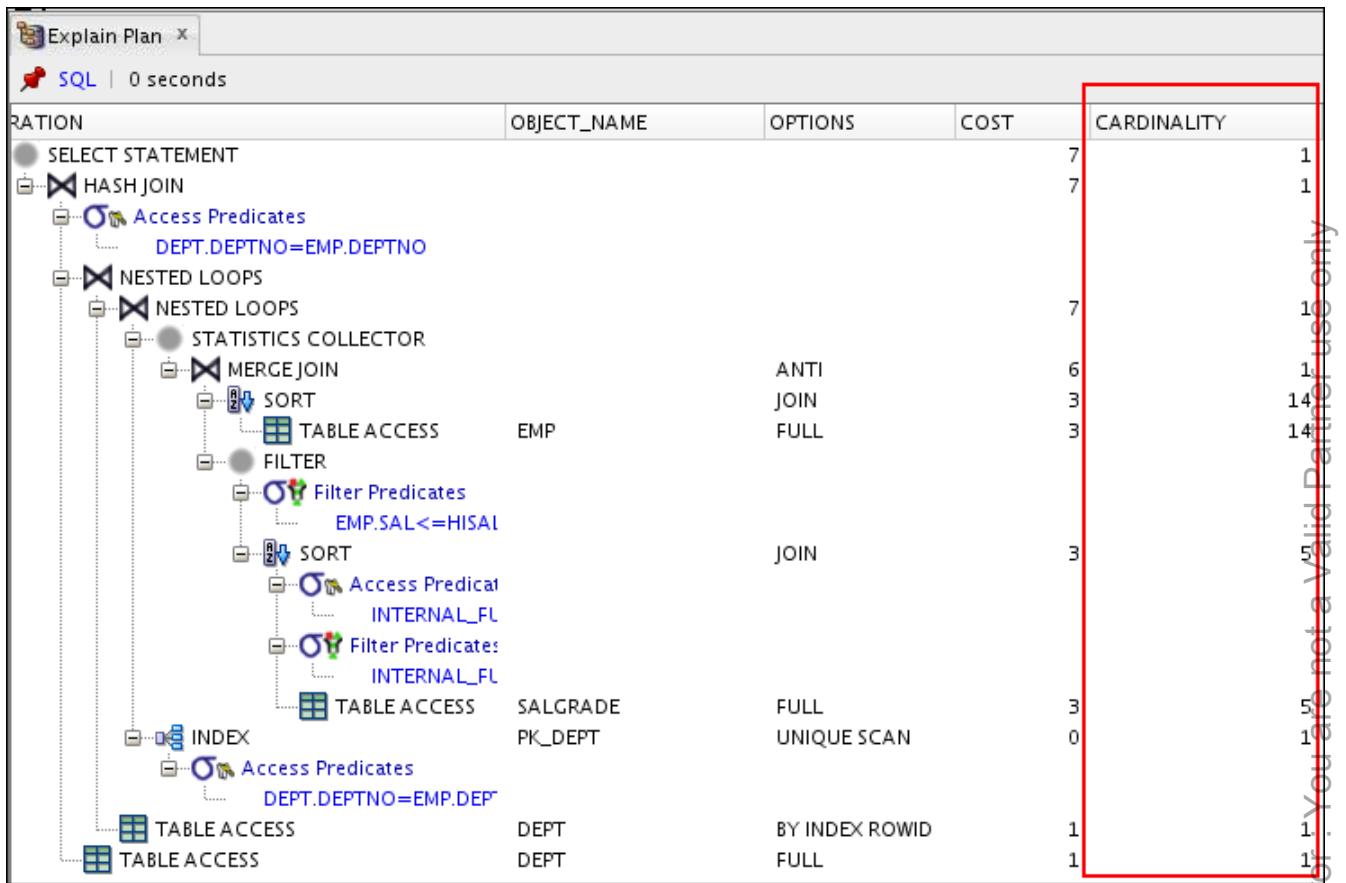
6. Customize the explain plan by adding Cardinality.
a. Select Tools > Preferences.



- b. Click Database > Autotrace/Explain Plan.
- c. Select Cardinality in the Explain Plan column and click OK.



The explain plan now displays Cardinality.



Oracle University and Ente... is not a valid Partition, use only 1

Practice 6-2: Extracting Execution Plans

In this practice, you use various methods to extract the execution plan used by the optimizer to execute a query. Note that all scripts needed for this practice can be found in your \$HOME/labs/solutions/Explain_Plan directory.

1. Connected as the oracle user from a terminal session, change the working directory to \$HOME/labs/solutions/Explain_Plan, and then execute the ep_startup.sh script. This script initializes the environment for this practice. A user called EP and a table called TEST have already been created that will be used throughout this practice.

```
$ cd /home/oracle/labs/solutions/Explain_Plan  
$ ./ep_startup.sh
```

```
[oracle@EDRSR19P1 Desktop]$ cd /home/oracle/labs/solutions/Explain_Plan/  
[oracle@EDRSR19P1 Explain_Plan]$ ./ep_startup.sh  
  
SQL*Plus: Release 12.1.0.1.0 Production on Thu Mar 21 23:36:25 2013  
  
Copyright (c) 1982, 2012, Oracle. All rights reserved.  
  
Last Successful login time: Wed Mar 13 2013 22:19:05 +00:00  
  
Connected to:  
Oracle Database 12c Enterprise Edition Release 12.1.0.1.0 - 64bit Production  
With the Partitioning, OLAP, Advanced Analytics and Real Application Testing options  
  
SQL>  
SQL> alter system flush shared_pool;  
  
System altered.  
  
SQL>  
SQL> alter system flush buffer_cache;  
  
System altered.  
  
SQL>  
SQL> set echo off  
Disconnected from Oracle Database 12c Enterprise Edition Release 12.1.0.1.0 - 64bit Production  
With the Partitioning, OLAP, Advanced Analytics and Real Application Testing options  
[oracle@EDRSR19P1 Explain_Plan]$ █
```

2. From the same terminal session (referred to as session 1 in the rest of this practice), be ready to execute the ep_session_issue.sh script. Enter the command, but do not execute it yet.

```
Session 1:  
-----  
  
$ ./ep_session_issue.sh
```

3. From a second terminal session (referred to as session 2 in the rest of this practice), connect as the oracle user. After this, connect to a SQL*Plus session as the SYS user. From this SQL*Plus session, be ready to use SQL Monitoring to monitor the execution plan used by session 1. You can execute the `ep_monitor.sql` script for this purpose. Enter the command, but do not execute it yet.

Note: Ensure that you understand the coordination between both sessions by pre-reading steps 4 and 5 before you continue.

```
Session 2:  
-----  
$ cd /home/oracle/labs/solutions/Explain_Plan  
  
$ sqlplus / as sysdba  
...  
SQL> @ep_monitor.sql
```

4. After you are ready in both the sessions, press Enter in session 1 to start the execution of the `ep_session_issue.sh` script. **Note:** Do not wait. Proceed with the next step immediately.

```
Session 1:  
-----  
$ ./ep_session_issue.sh  
  
[oracle@EDRSR19P1 Explain_Plan]$ ./ep_session_issue.sh  
SQL*Plus: Release 12.1.0.1.0 Production on Fri Mar 22 01:48:34 2013  
Copyright (c) 1982, 2012, Oracle. All rights reserved.  
Last Successful login time: Fri Mar 22 2013 01:45:31 +00:00  
  
Connected to:  
Oracle Database 12c Enterprise Edition Release 12.1.0.1.0 - 64bit Production  
With the Partitioning, OLAP, Advanced Analytics and Real Application Testing options
```

5. In session 2, press the Enter key to start the execution of the `ep_monitor.sql` script. After the execution, enter "/" and go back to your SQL*Plus session as many times as necessary until session 1 is finished with its execution. What do you observe?
You can see that session 1 uses NESTED LOOPS on top of two INDEX RANGE SCANS to execute the query. It takes approximately 47 seconds to execute session 1's query. The time depends on your environment. The big advantage of SQL Monitoring is that you can clearly see which steps in the execution plan take up most of the resources. In this case, you clearly see that you do only one scan of the index, and that for each row returned, you execute another index scan to probe. This is not really efficient. Also, there is no cost information for this monitored plan.

```
Session 2:  
-----  
SQL> @ep_monitor.sql
```

```
SQL> @ep_monitor.sql  
SQL> set long 100000000  
SQL> set longchunksize 10000000  
SQL> set linesize 200  
SQL> set pagesize 1000  
SQL>  
SQL> exec dbms_lock.sleep(8);  
  
PL/SQL procedure successfully completed.  
  
SQL>  
SQL> select dbms_sqltune.report_sql_monitor(sql_id=>'dkz7v96ym42c6', report_level=>'ALL') from dual;  
DBMS_SQLTUNE.REPORT_SQL_MONITOR(SQL_ID=>'DKZ7V96YM42C6', REPORT_LEVEL=>'ALL')  
-----  
SQL Monitoring Report  
  
SQL Text  
-----  
select count(*) from test t1, test t2 where t1.c=t2.c and t1.c=1  
  
Global Information  
-----  
Status : EXECUTING  
Instance ID : 1  
Session : EP (35:8379)  
SQL ID : dkz7v96ym42c6  
SQL Execution ID : 16777217  
Execution Started : 03/22/2013 01:48:34  
First Refresh Time : 03/22/2013 01:48:40  
Last Refresh Time : 03/22/2013 01:48:44  
Duration : 11s  
Module/Action : SQL*Plus/-  
Service : SYS$USERS  
Program : sqlplus@EDRSR19P1 (TNS V1-V3)
```

```

Global Stats
=====
| Elapsed | Cpu | IO | Other | Buffer | Read | Read |
| Time(s) | Time(s) | Waits(s) | Waits(s) | Gets | Reqs | Bytes |
| 10 | 9.31 | 0.00 | 0.28 | 285K | 40 | 320KB |

SQL Plan Monitoring Details (Plan Hash Value=1643938535)
=====
| Id | Operation | Name | Rows | Cost | Time | Start | Execs | Rows | Read | Read | Activity | Activity Detail |
| | | | (Estim) | | Active(s) | Active | | (Actual) | Reqs | Bytes | (%) | (# samples) |
| -> 0 | SELECT STATEMENT | | | | 5 | +6 | 1 | 0 | | | |
| -> 1 | SORT AGGREGATE | | | | 5 | +6 | 1 | 0 | | | |
| -> 2 | NESTED LOOPS | | | | 5 | +6 | 1 | 146M | | | |
| -> 3 | INDEX RANGE SCAN | TEST_C_INDX | | | 5 | +6 | 1 | 7297 | 1 | 8192 | | |
| -> 4 | INDEX RANGE SCAN | TEST_C_INDX | | | 11 | +0 | 7298 | 146M | 1 | 8192 | 100.00 | Cpu (11)

SQL>
SQL> /
DBMS_SQLTUNE.REPORT_SQL_MONITOR(SQL_ID=>'DKZ7V96YM42C6',REPORT_LEVEL=>'ALL')
-----
SQL Monitoring Report
SQL Text
-----
select count(*) from test t1, test t2 where t1.c=t2.c and t1.c=1
Global Information
-----
Status : DONE (ALL ROWS)
Instance ID : 1
Session : EP (35:8379)
SQL ID : dkz7v96ym42c6
SQL Execution ID : 16777217
Execution Started : 03/22/2013 01:48:34
First Refresh Time : 03/22/2013 01:48:40
Last Refresh Time : 03/22/2013 01:49:00
Duration : 26s
Module/Action : SQL*Plus/-
Service : SYS$USERS
Program : sqlplus@EDRSR19P1 (TNS V1-V3)
Fetch Calls : 1

```

After 30–50 seconds (depending on your environment), you should see the following output in your session 1:

Session 1:

```

SQL>
SQL> set timing on
SQL>
SQL> select count(*) from test t1, test t2 where t1.c=t2.c and t1.c=1;

COUNT(*)
-----
4000000000

Elapsed: 00:00:26.33
SQL>
SQL> exit;

```

6. From session 1, connect as the EP user in the SQL*Plus session.

```
Session 1:  
-----  
$ sqlplus ep/ep  
...  
SQL>
```

7. Use PLAN_TABLE to determine the execution plan of the query that was executed in step 4. Run the ep_explain.sql script. What do you observe?

This time, the execution plan uses a hash join on top of two index fast full scans.

```
SQL> @ep_explain.sql  
SQL>  
SQL> set linesize 200 pagesize 1000  
SQL>  
SQL> explain plan for  
2 select count(*) from test t1, test t2 where t1.c=t2.c and t1.c=1;  
  
Explained.  
  
SQL>  
SQL> select * from table(dbms_xplan.display);  
  
PLAN_TABLE_OUTPUT  
-----  
Plan hash value: 3253233075  
  
| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time |  
|---|---|---|---|---|---|---|  
| 0 | SELECT STATEMENT | | 1 | 6 | 1022 (98) | 00:00:01 |  
| 1 | SORT AGGREGATE | | 1 | 6 | 1022 (98) | 00:00:01 |  
/* 2 | HASH JOIN | | 400M | 2288M | 1022 (98) | 00:00:01 |  
/* 3 | INDEX FAST FULL SCAN| TEST_C_INDX | 20000 | 60000 | 12 (0) | 00:00:01 |  
/* 4 | INDEX FAST FULL SCAN| TEST_C_INDX | 20000 | 60000 | 12 (0) | 00:00:01 |  
  
Predicate Information (identified by operation id):  
-----  
2 - access("T1"."C"="T2"."C")  
3 - filter("T1"."C"=1)  
4 - filter("T2"."C"=1)  
  
18 rows selected.
```

8. Now, you want to monitor this execution plan that uses a hash join to compare it with the one generated in step 4. In addition, you want to make sure that you use the correct plan this time. So, in your session 1, start Autotrace and be ready to execute the following query. Do not execute it yet because you need to start SQL Monitoring in your session 2:

```
Session 1:  
-----  
SQL> set autotrace on  
SQL> @ep_execute
```

9. From your session 2, be ready to execute the SQL Monitoring command again. Do not execute it yet, though.

```
Session 2:  
-----  
SQL> @ep_monitor.sql
```

10. Start the execution of your query from session 1 by pressing Enter.

Note: Move to the next step without waiting.

```
Session 1:  
-----  
SQL> @ep_execute
```

```
SQL>  
SQL> set autotrace on  
SQL> @ep_execute  
SQL> set echo on  
SQL>  
SQL> set timing on  
SQL>  
SQL> select count(*) from test t1, test t2 where t1.c=t2.c and t1.c=1;
```

11. From session 2, start monitoring your query by pressing Enter. After the query is executed, enter "/" and go back to your SQL*Plus session as many times as necessary until session 1 is finished with its execution. What do you observe?

You can see that the optimizer uses a hash join on top of two index fast full scans. Looking at the various reports, you can clearly see how the optimizer processes a hash join by reading the driving index in memory first. This operation is quick. Though you cannot see it run, it is already done the first time you look at it. Then the probe is performed on the index again. This operation takes more time. Also, note that cost information is provided in the execution plan.

```
Session 2:
```

```
-----
```

```
SQL> @ep_monitor.sql
SQL> set echo on
```

```
SQL> @ep_monitor.sql
SQL> set echo on
SQL> set long 10000000
SQL> set longchunksize 10000000
SQL> set linesize 200
SQL> set pagesize 1000
SQL>
SQL> exec dbms_lock.sleep(8);

PL/SQL procedure successfully completed.

SQL>
SQL> select dbms_sqltune.report_sql_monitor(sql_id=>'dkz7v96ym42c6', report_level=>'ALL') from dual;

DBMS_SQLTUNE.REPORT_SQL_MONITOR(SQL_ID=>'DKZ7V96YM42C6', REPORT_LEVEL=>'ALL')
-----
SQL Monitoring Report

SQL Text
-----
select count(*) from test t1, test t2 where t1.c=t2.c and t1.c=1

Global Information
-----
Status          : EXECUTING
Instance ID     : 1
Session          : EP (35:8403)
SQL ID          : dkz7v96ym42c6
SQL Execution ID: 16777218
Execution Started: 03/22/2013 01:55:58
First Refresh Time: 03/22/2013 01:56:04
Last Refresh Time: 03/22/2013 01:56:06
Duration        : 11s
Module/Action   : SQL*Plus/-
Service          : SYS$USERS
Program         : sqlplus@EDRSR19P1 (TNS V1-V3)
```

```

Global Stats
-----
| Elapsed | Cpu | IO | Other | Buffer | Read | Read |
| Time(s) | Time(s) | Waits(s) | Waits(s) | Gets | Reqs | Bytes |
| 7.97 | 7.82 | 0.00 | 0.14 | 59 | 2 | 16384 |

SQL Plan Monitoring Details (Plan Hash Value=3253233075)
-----
| Id | Operation | Name | Rows | Cost | Time | Start | Execs | Rows | Read | Read | Mem | Activity | Activity Detail |
| | | | (Estim) | | Active(s) | Active | | (Actual) | Reqs | Bytes | | (%) | | (# samples) |
| -> 0 | SELECT STATEMENT | | | | 3 | +6 | 1 | 0 | | | | | |
| -> 1 | SORT AGGREGATE | | 1 | | 3 | +6 | 1 | 0 | | | | | |
| -> 2 | HASH JOIN | 400M | 1022 | 11 | +1 | 1 | 72M | | | | 2M | 100.00 | Cpu (11) |
| 3 | INDEX FAST FULL SCAN | TEST_C_INDX | 20000 | 12 | 1 | +6 | 1 | 20000 | 1 | 8192 | | | |
| -> 4 | INDEX FAST FULL SCAN | TEST_C_INDX | 20000 | 12 | 3 | +6 | 1 | 3584 | | | | | |

SQL>
SQL> /
DBMS_SQLTUNE.REPORT_SQL_MONITOR(SQL_ID=>'DKZ7V96YM42C6',REPORT_LEVEL=>'ALL')
-----
SQL Monitoring Report
SQL Text
-----
select count(*) from test t1, test t2 where t1.c=t2.c and t1.c=1

Global Information
-----
Status : EXECUTING
Instance ID : 1
Session : EP (35:8403)
SQL ID : dkz7v96ym42c6
SQL Execution ID : 16777218
Execution Started : 03/22/2013 01:55:58
First Refresh Time : 03/22/2013 01:56:04
Last Refresh Time : 03/22/2013 01:56:09
Duration : 13s
Module/Action : SQL*Plus/-
Service : SYS$USERS
Program : sqlplus@EDRSR19P1 (TNS V1-V3)

```

```

SQL> /
DBMS_SQLTUNE.REPORT_SQL_MONITOR(SQL_ID=>'DKZ7V96YM42C6',REPORT_LEVEL=>'ALL')
-----
SQL Monitoring Report
SQL Text
-----
select count(*) from test t1, test t2 where t1.c=t2.c and t1.c=1

Global Information
-----
Status : EXECUTING
Instance ID : 1
Session : EP (35:8403)
SQL ID : dkz7v96ym42c6
SQL Execution ID : 16777218
Execution Started : 03/22/2013 01:55:58
First Refresh Time : 03/22/2013 01:56:04
Last Refresh Time : 03/22/2013 01:56:09
Duration : 13s
Module/Action : SQL*Plus/-
Service : SYS$USERS
Program : sqlplus@EDRSR19P1 (TNS V1-V3)

Global Stats
-----
| Elapsed | Cpu | IO | Other | Buffer | Read | Read |
| Time(s) | Time(s) | Waits(s) | Waits(s) | Gets | Reqs | Bytes |
| 10 | 10 | 0.00 | 0.33 | 61 | 2 | 16384 |

SQL Plan Monitoring Details (Plan Hash Value=3253233075)
-----
| Id | Operation | Name | Rows | Cost | Time | Start | Execs | Rows | Read | Read | Mem | Activity | Activity Detail |
| | | | (Estim) | | Active(s) | Active | | (Actual) | Reqs | Bytes | | (%) | | (# samples) |
| -> 0 | SELECT STATEMENT | | | | 6 | +6 | 1 | 0 | | | | | |
| -> 1 | SORT AGGREGATE | | 1 | | 6 | +6 | 1 | 0 | | | | | |
| -> 2 | HASH JOIN | 400M | 1022 | 13 | +1 | 1 | 92M | | | | 2M | 100.00 | Cpu (13) |
| 3 | INDEX FAST FULL SCAN | TEST_C_INDX | 20000 | 12 | 1 | +6 | 1 | 20000 | 1 | 8192 | | | |
| -> 4 | INDEX FAST FULL SCAN | TEST_C_INDX | 20000 | 12 | 6 | +6 | 1 | 4608 | | | | | |


```

12. When your query is executed, what do you observe in your session 1?

Session 1 also reports the same execution plan as the one you observed in session 2.

```
SQL> select count(*) from test t1, test t2 where t1.c=t2.c and t1.c=1;

COUNT(*)
-----
4000000000

Elapsed: 00:00:45.26

Execution Plan
-----
Plan hash value: 3253233075

| Id  | Operation          | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
|---|---|---|---|---|---|---|
| 0  | SELECT STATEMENT   |           |       1 |       6 | 1022  (98)| 00:00:01 |
| 1  | SORT AGGREGATE    |           |       1 |       6 |            |          |
|* 2  | HASH JOIN          |           | 400M  | 2288M | 1022  (98)| 00:00:01 |
|* 3  | INDEX FAST FULL SCAN| TEST_C_INDX | 20000 | 60000 |    12  (0) | 00:00:01 |
|* 4  | INDEX FAST FULL SCAN| TEST_C_INDX | 20000 | 60000 |    12  (0) | 00:00:01 |

Predicate Information (identified by operation id):
-----
2 - access("T1"."C"="T2"."C")
3 - filter("T1"."C"=1)
4 - filter("T2"."C"=1)

Statistics
-----
0 recursive calls
0 db block gets
92 consistent gets
2 physical reads
0 redo size
542 bytes sent via SQL*Net to client
543 bytes received via SQL*Net from client
2 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)
1 rows processed
```

13. In session 1, disable Autotrace.

```
Session 1:
-----
SQL> set autotrace off
SQL>
```

14. From session 1, how can you ensure that you gather all execution plan statistics for the following query without changing any session parameters? Implement your solution.

```
select count(*) from test t1, test t2 where t1.c=t2.c and  
t1.c=1;
```

Session 1:

```
SQL> @ep_execute_with_all
```

```
SQL> @ep_execute_with_all  
SQL> set echo on  
SQL>  
SQL> set timing on  
SQL>  
SQL> select /*+ gather_plan_statistics */ count(*) from test t1, test t2 where t  
1.c=t2.c and t1.c=1;  
  
COUNT(*)  
-----  
4000000000  
  
Elapsed: 00:00:30.48  
SQL>
```

15. From session 1, retrieve all execution plans corresponding to all the queries that you executed since the beginning of this practice. What is your conclusion?

- a. The easiest way to find all the plans is to look at the content of the SGA by using the dbms_xplan.display_cursor function. First, you must determine the SQL_IDs used to represent your queries. You essentially have two queries, and one that has two children. You should now understand what happened in step 4. There was no cost information due to the use of the rule-based optimizer instead of the cost-based one.

Session 1:

```
SQL> @ep_retrieve_all_plans  
SQL> set echo on  
SQL>  
SQL> set linesize 200 pagesize 1000  
SQL>  
SQL> col sql_text format a50  
SQL>  
SQL> select sql_id,plan_hash_value,sql_text from v$sql where sql_text  
like '%from test t1, test t2%';  
  
SQL_ID          PLAN_HASH_VALUE SQL_TEXT  
-----  
-----  
dkz7v96ym42c6      3253233075 select count(*) from test t1, test t2  
where t1.c=t  
2.c and t1.c=1
```

```
dkz7v96ym42c6      1643938535 select count(*) from test t1, test t2
where t1.c=t
                           2.c and t1.c=1

8w580dd6ncgqw      3253233075 select /*+ gather_plan_statistics */
count(*) from
                           test t1, test t2 where t1.c=t2.c and
t1.c=1

0w0va2d7hhtxa      3253233075 explain plan for select count(*) from
test t1, tes
                           t t2 where t1.c=t2.c and t1.c=1

dd09kf5dnplgt      903671040 select sql_id,plan_hash_value,sql_text
from v$sql
                           where sql_text like '%from test t1, test
t2%'

0tu97rt51pjwv      3253233075 EXPLAIN PLAN SET
STATEMENT_ID='PLUS760049' FOR sel
                           ect count(*) from test t1, test t2 where
t1.c=t2.c
                           and t1.c=1

6 rows selected.

Elapsed: 00:00:00.03
SQL>
SQL> select * from
table(dbms_xplan.display_cursor('dkz7v96ym42c6',null,'TYPICAL'));

PLAN_TABLE_OUTPUT
-----
-----
-----
SQL_ID  dkz7v96ym42c6, child number 0
-----
select count(*) from test t1, test t2 where t1.c=t2.c and t1.c=1

Plan hash value: 3253233075

-----
| Id  | Operation          | Name        | Rows  | Bytes | Cost
(%CPU) | Time       |
-----
```

```

|   0 | SELECT STATEMENT          |           |           |           | 1261
(100)|           |
|   1 |   SORT AGGREGATE         |           |           |   1 |       6 |
|           |
|*  2 |     HASH JOIN            |           |           | 400M| 2288M| 1261
(99)| 00:00:16 |
|*  3 |       INDEX FAST FULL SCAN| TEST_C_INDX | 20000 | 60000 |      12
(0) | 00:00:01 |
|*  4 |       INDEX FAST FULL SCAN| TEST_C_INDX | 20000 | 60000 |      12
(0) | 00:00:01 |
-----  

-----  

Predicate Information (identified by operation id):  

-----  

2 - access("T1"."C"="T2"."C")
3 - filter("T1"."C"=1)
4 - filter("T2"."C"=1)

SQL_ID dkz7v96ym42c6, child number 1
-----  

select count(*) from test t1, test t2 where t1.c=t2.c and t1.c=1

Plan hash value: 1643938535

-----  

| Id  | Operation          | Name    |  

-----  

|   0 | SELECT STATEMENT  |          |  

|   1 |   SORT AGGREGATE |          |  

|   2 |   NESTED LOOPS   |          |  

|*  3 |     INDEX RANGE SCAN| TEST_C_INDX |  

|*  4 |     INDEX RANGE SCAN| TEST_C_INDX |  

-----  

-----  

Predicate Information (identified by operation id):  

-----  

3 - access("T1"."C"=1)
4 - access("T1"."C"="T2"."C")

Note
-----
- rule based optimizer used (consider using cbo)
```

```
49 rows selected.

Elapsed: 00:00:00.09
SQL>
SQL> select * from
table(dbms_xplan.display_cursor('8w580dd6ncgqw',null,'ADVANCED
ALLSTATS LAST'));

PLAN_TABLE_OUTPUT
-----
-----
-----
SQL_ID  8w580dd6ncgqw, child number 0
-----
select /*+ gather_plan_statistics */ count(*) from test t1, test t2
where t1.c=t2.c and t1.c=1

Plan hash value: 3253233075

-----
-----
-----
| Id  | Operation          | Name   | Starts | E-Rows | E-
Bytes| Cost (%CPU) | E-Time   | A-Rows | A-Time | Buffers | OMem |
1Mem | Used-Mem |           |
-----
| 0  | SELECT STATEMENT |        | 1 | 00:01:05.38 | 1 | 92 | - |
1261 (100)|           |        | 1 | 00:01:05.38 | 1 | 92 | - |
| 1  | SORT AGGREGATE |        | 1 | 00:01:05.38 | 1 | 92 | 1 | 6 |
| * 2 | HASH JOIN        |        | 400M | 00:04:43.33 | 1 | 92 | 400M |
2288M| 1261 (99) | 00:00:16 | 1098K (0) |           | 1 | 92 | 1155K |
1155K|           |           |           |           |           |           |
| * 3 | INDEX FAST FULL SCAN| TEST_C_INDX | 1 | 20000 | 60000 |
12 (0) | 00:00:01 | 20000 | 00:00:00.03 | 1 | 20000 | 60000 |
| * 4 | INDEX FAST FULL SCAN| TEST_C_INDX | 1 | 20000 | 60000 |
12 (0) | 00:00:01 | 20000 | 00:00:00.18 | 1 | 20000 | 60000 |
|           |           |           |           |           |           |           |
-----
```

Query Block Name / Object Alias (identified by operation id):

```
1 - SEL$1
3 - SEL$1 / T1@SEL$1
4 - SEL$1 / T2@SEL$1

Outline Data
-----
/*+
BEGIN_OUTLINE_DATA
IGNORE_OPTIM_EMBEDDED_HINTS
OPTIMIZER_FEATURES_ENABLE('11.2.0.1')
DB_VERSION('11.2.0.1')
ALL_ROWS
OUTLINE_LEAF(@"SEL$1")
INDEX_FFS(@"SEL$1" "T1"@"SEL$1" ("TEST"."C"))
INDEX_FFS(@"SEL$1" "T2"@"SEL$1" ("TEST"."C"))
LEADING(@"SEL$1" "T1"@"SEL$1" "T2"@"SEL$1")
USE_HASH(@"SEL$1" "T2"@"SEL$1")
END_OUTLINE_DATA
*/
Predicate Information (identified by operation id):
-----
2 - access("T1"."C"="T2"."C")
3 - filter("T1"."C"=1)
4 - filter("T2"."C"=1)

Column Projection Information (identified by operation id):
-----
1 - (#keys=0) COUNT(*) [22]
2 - (#keys=1)
3 - "T1"."C" [NUMBER, 22]
4 - "T2"."C" [NUMBER, 22]

56 rows selected.

Elapsed: 00:00:00.18
SQL>
SQL>
```

16. From session 1, try to retrieve your execution plans from the Automatic Workload Repository. What happens and why?
- You can use the previously found SQL_IDS to search through the DBA_HIST_SQLTEXT view. You should see that right now, none of your queries are stored in AWR.
Note: It is possible that a snapshot was taken during this practice. If so, some or all of your queries are stored in AWR.

```
Session 1:
```

```
-----
```

```
SQL> @ep_retrieve_awr
```

```
SQL> @ep_retrieve_awr
SQL> set echo on
SQL>
SQL> set linesize 200
SQL>
SQL> SELECT SQL_ID, SQL_TEXT FROM dba_hist_sqltext
  2 WHERE SQL_ID in ('dkz7v96ym42c6','8w580dd6ncgqw');

SQL_ID
-----
SQL_TEXT
-----
dkz7v96ym42c6
select count(*) from test t1, test t2 where t1.c=t2.c and t1.c=1

8w580dd6ncgqw
select /*+ gather_plan_statistics */ count(*) from test t1, test t2 where t1.c=t2.c and t1.c=1

SQL>
```

17. How can you ensure that you retrieve your queries from the Automatic Workload Repository? Implement your solution.
- You must flush the SGA information to AWR. You can use DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT for this purpose.

```
Session 1:
```

```
-----
```

```
SQL> @ep_save_awr
```

```
SQL>
SQL> @ep_save_awr
SQL> set echo on
SQL>
SQL> EXEC DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT('ALL');

PL/SQL procedure successfully completed.
```

18. Verify that your solution works.

Session 1:

SQL> @ep_show_awr

```
SQL> @ep_show_awr
SQL> set echo on
SQL>
SQL> set linesize 200 pagesize 1000
SQL>
SQL> SELECT PLAN_TABLE_OUTPUT
  2  FROM
  3  TABLE (DBMS_XPLAN.DISPLAY_AWR('dkz7v96ym42c6'));

PLAN_TABLE_OUTPUT
-----
SQL_ID dkz7v96ym42c6
-----
select count(*) from test t1, test t2 where t1.c=t2.c and t1.c=1

Plan hash value: 1643938535

| Id | Operation          | Name      |
|---|---|
| 0 | SELECT STATEMENT   |
| 1 |   SORT AGGREGATE    |
| 2 |     NESTED LOOPS    |
| 3 |       INDEX RANGE SCAN| TEST_C_INDX |
| 4 |       INDEX RANGE SCAN| TEST_C_INDX |

Note
-----
- rule based optimizer used (consider using cbo)

SQL_ID dkz7v96ym42c6
-----
select count(*) from test t1, test t2 where t1.c=t2.c and t1.c=1

Plan hash value: 3253233075

| Id | Operation          | Name      | Rows  | Bytes | Cost (%CPU)| Time      |
|---|---|
| 0 | SELECT STATEMENT   |
| 1 |   SORT AGGREGATE    |
| 2 |     HASH JOIN        |
| 3 |       INDEX FAST FULL SCAN| TEST_C_INDX | 20000 | 60000 | 1015  (98) | 00:00:01 |
| 4 |       INDEX FAST FULL SCAN| TEST_C_INDX | 20000 | 60000 | 12    (0)  | 00:00:01 |

36 rows selected.
```

```

SQL>
SQL> SELECT PLAN_TABLE_OUTPUT
  2  FROM
  3  TABLE (DBMS_XPLAN.DISPLAY_AWR('8w580dd6ncgqw',null,null,'TYPICAL ALLSTATS LAST'));

PLAN_TABLE_OUTPUT
-----
SQL_ID 8w580dd6ncgqw
-----
select /*+ gather_plan_statistics */ count(*) from test t1, test t2
where t1.c=t2.c and t1.c=1

Plan hash value: 3253233075

| Id | Operation          | Name      | E-Rows | E-Bytes | Cost (%CPU) | E-Time   |
|---|---|---|---|---|---|---|
| 0 | SELECT STATEMENT   |           |         |         | 1022 (100)  |          |
| 1 |   SORT AGGREGATE    |           |       1 |        6 |            |          |
| 2 |   HASH JOIN          |           | 400M  | 2288M  | 1022 (98)  | 00:00:01  |
| 3 |     INDEX FAST FULL SCAN| TEST_C_INDX | 20000 | 60000  | 12 (0)    | 00:00:01  |
| 4 |     INDEX FAST FULL SCAN| TEST_C_INDX | 20000 | 60000  | 12 (0)    | 00:00:01  |

Note
-----
- Warning: basic plan statistics not available. These are only collected when:
  * hint 'gather_plan_statistics' is used for the statement or
  * parameter 'statistics_level' is set to 'ALL', at session or system level

23 rows selected.

```

19. Exit from both SQL*Plus sessions.

```

Session 1:
-----
SQL> exit
Disconnected ...
$
```

Do not forget to exit from session 2:

```

Session 2:
-----
SQL> exit
Disconnected ...
$
```

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Error : You are not a Valid Partner use only

Practices for Lesson 7: Interpreting Execution Plans and Enhancements

Chapter 7

Practices for Lesson 7: Overview

Practices Overview

This practice covers the dynamic plans part of the Adaptive Execution Plans feature in Oracle Database 12c.

Practice 7-1: Using Dynamic Plans

Overview

In this practice, you use the dynamic plans part of the Adaptive Execution Plans feature.

Tasks

- Set the .oraenv environment and start sqlplus as SYS.

```
$ . oraenv  
ORACLE_SID = [oracle] ? sysun  
The Oracle base remains unchanged with value /u01/app/oracle  
$ sqlplus / as sysdba
```

- Grant the SELECT ANY DICTIONARY privilege to OE.

```
grant select any dictionary to oe;
```

```
[oracle@EDT3R13P1 Explain_Plan]$ sqlplus / as sysdba  
SQL*Plus: Release 12.1.0.1.0 Production on Wed Sep 18 07:17:59 2013  
Copyright (c) 1982, 2013, Oracle. All rights reserved.  
  
Connected to:  
Oracle Database 12c Enterprise Edition Release 12.1.0.1.0 - 64bit Production  
With the Partitioning, OLAP, Advanced Analytics and Real Application Testing options  
  
SQL> grant select any dictionary to oe;  
  
Grant succeeded.  
  
SQL> ■
```

- From the same SQL*Plus session, connect as the OE user and show the execution plan of the following query without executing it:

```
select /*+ monitor*/ product_name  
from order_items o, product_information p  
where o.unit_price = 15  
and quantity > 1  
and p.product_id = o.product_id;
```

- Connect as oe with oe as the password.
- Execute the following query:

```
EXPLAIN PLAN FOR  
select /*+ monitor*/ product_name  
from order_items o, product_information p  
where o.unit_price = 15  
and quantity > 1  
and p.product_id = o.product_id;
```

```
SQL> set linesize 132
SQL> select * from table(dbms_xplan.display());
```

```
SQL> connect oe/oe
Connected.
SQL> explain plan for
select /*+ monitor*/ product_name
from order_items o, product_information p
where o.unit_price = 15
and quantity > 1
and p.product_id = o.product_id;
2   3   4   5   6
Explained.

SQL> set linesize 300
SQL> select * from table(dbms_xplan.display());
```

| PLAN TABLE OUTPUT | | | | | | | | |
|-------------------|---------------------|-----------------------------|------------------------|-------|-------------|----------|----------|--|
| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time | | |
| 0 | SELECT STATEMENT | | 4 | 128 | 7 (0) | 00:00:01 | | |
| 1 | <u>NESTED LOOPS</u> | | | | | | | |
| 2 | NESTED LOOPS | | 4 | 128 | 7 (0) | 00:00:01 | | |
| * | 3 | TABLE ACCESS FULL | ORDER_ITEMS | 4 | 48 | 3 (0) | 00:00:01 | |
| * | 4 | INDEX UNIQUE SCAN | PRODUCT_INFORMATION_PK | 1 | | 0 (0) | 00:00:01 | |
| | 5 | TABLE ACCESS BY INDEX ROWID | PRODUCT_INFORMATION | 1 | 20 | 1 (0) | 00:00:01 | |

PLAN_TABLE_OUTPUT

Predicate Information (identified by operation id):

```
3 - filter("O"."UNIT_PRICE"=15 AND "QUANTITY">>1)
4 - access("P"."PRODUCT_ID"="O"."PRODUCT_ID")
```

22 rows selected.

4. What do you observe?

The plan is using a simple NESTED LOOPS join.

5. Now, execute the same query, as follows:

```
select /*+ monitor*/ product_name
```

```
from order_items o, product_information p
where o.unit_price = 15
and quantity > 1
and p.product_id = o.product_id;
```

```
SQL> select /*+ monitor*/ product_name
  from order_items o, product_information p
 where o.unit_price = 15
   and quantity > 1
   and p.product_id = o.product_id;
  2      3      4      5      6
PRODUCT_NAME
-----
Screws <B.28.S>
Screws <B.28.S>
Screws <B.28.S>
Screws <B.28.S>
...
Screws <B.28.S>
Screws <B.28.S>

13 rows selected.

SQL>
```

6. Show the resulting execution plan by using the following command:

```
select * from table(dbms_xplan.display_cursor());
```

```

SQL> select * from table(dbms_xplan.display_cursor());

PLAN_TABLE_OUTPUT
-----
SQL_ID 9ht2704s2s5s9, child number 1

SELECT product_name FROM order_items o, product_information p WHERE
o.unit_price = 15 AND quantity > 1 AND p.product_id = o.product_id

Plan hash value: 1553478007

-----
| Id  | Operation          | Name           | Rows | Bytes | Cost (%CPU) |
-----| 0  | SELECT STATEMENT   |                |      |       | 8 (100) |
|   - |                         |
| * 1 | HASH JOIN          |                | 13   | 416   | 8   (0) |
| 00:00:01 |                         |

PLAN_TABLE_OUTPUT
-----
|* 2 | TABLE ACCESS FULL| ORDER_ITEMS    | 13  | 156  | 3   (0) |
| 00:00:01 |                         |

| 3 | TABLE ACCESS FULL| PRODUCT_INFORMATION | 288 | 5760 | 5   (0) |
| 00:00:01 |                         |

-----
Predicate Information (identified by operation id):
PLAN_TABLE_OUTPUT
-----
1 - access("P"."PRODUCT_ID"="O"."PRODUCT_ID")
2 - filter(("O"."UNIT_PRICE"=15 AND "QUANTITY">>1))

Note
-----
- statistics feedback used for this statement
- this is an adaptive plan

```

7. What do you observe and conclude?

Answer: The actual plan used at execution was a HASH_JOIN.

Why did the plan change?

The plan changed because the optimizer realized during execution that the number of rows actually returned from the `order_items` table was much larger than expected. Multiple single-column predicates on the `order_items` table caused the initial cardinality estimate to be

incorrect. The mis-estimation cannot be corrected by extended statistics because one of the predicates is a non-equality predicate.

8. How would you confirm whether the plan change was caused by dynamic plans?

By looking in V\$SQL and checking the value of the new column

IS_RESOLVED_ADAPTIVE_PLAN

```
SQL> column sql_text format a30
SQL> select sql_id, sql_text, is_resolved_adaptive_plan
  from v$sql
 where sql_text like 'select /*+ monitor*/ product_name%';
```

```
SQL> column sql_text format a30
SQL> select sql_id, sql_text, is_resolved_adaptive_plan
  from v$sql
 where sql_text like 'select /*+ monitor*/ product_name%';
 2   3
SQL_ID      SQL_TEXT          I
-----
9ht2704s2s5s9 select /*+ monitor*/ product_n Y
ame from order_items o, produc
t_information p where o.unit_p
rice = 15 and quantity > 1 and
p.product_id = o.product_id
```

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Error : You are not a Valid Partner use only

Practices for Lesson 8: Optimizer: Table and Index Access Paths

Chapter 8

Practices for Lesson 8: Overview

Practices Overview

In these practices, you will examine the access paths chosen by the optimizer for 12 different cases of table and index access.

List of Cases:

- Case 1: With and Without Index
- Case 2: Compare Single Column Index Access
- Case 3: Concatenated Index
- Case 4: Bitmap Index Access
- Case 5: Complex Predicate with Bitmap Indexes
- Case 6: Index Only Access
- Case 7: Index Join
- Case 8: Bitmap Index Only Access
- Case 9: B*-Tree Index Only Access
- Case 10: Function Based Index
- Case 11: Index Organized Table
- Case 12: Index Skip Scan

Practice 8-1: Using Different Access Paths

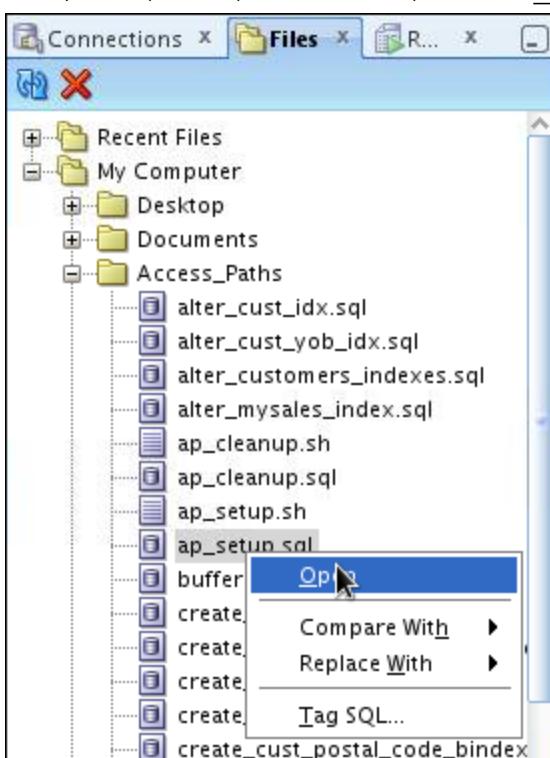
Overview

In this practice, you explore various access paths that the optimizer can use, and compare them. You have the possibility of exploring 12 different scenarios, each of which is self-contained. All the scripts needed for this practice can be found in the `$HOME/labs/solutions/Access_Paths` directory. It is helpful to set the worksheet preferences to use `/home/oracle/labs/solutions/Access_Paths` as the default lookup path for scripts.

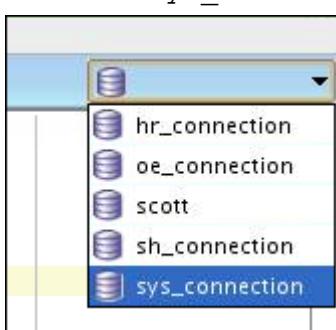
Tasks

1. Case 1: With and Without Index

- Use the SQL Developer Files tab and open the `$HOME/labs/solutions/Access_Paths/ap_setup.sql` script.



- Select `sys_connection`.



- Execute the script (F5).

The screenshot shows the Oracle SQL Developer interface. The title bar says "ap_setup.sql". The main area is a "Worksheet" tab with the following SQL code:

```
--Execute as the sys user
grant dba to sh;
exit;
```

Below the worksheet is a "Script Output" window showing the results of the execution:

```
grant succeeded.
Commit
```

The total execution time is listed as 0.35299999 seconds.

2. Open idx_setup.sql and execute it with sh_connection.

The screenshot shows the Oracle SQL Developer interface with two tabs open: "ap_setup.sql" and "idx_setup.sql". The "idx_setup.sql" tab is active and shows the following SQL code:

```
set echo on

drop table mysales purge;

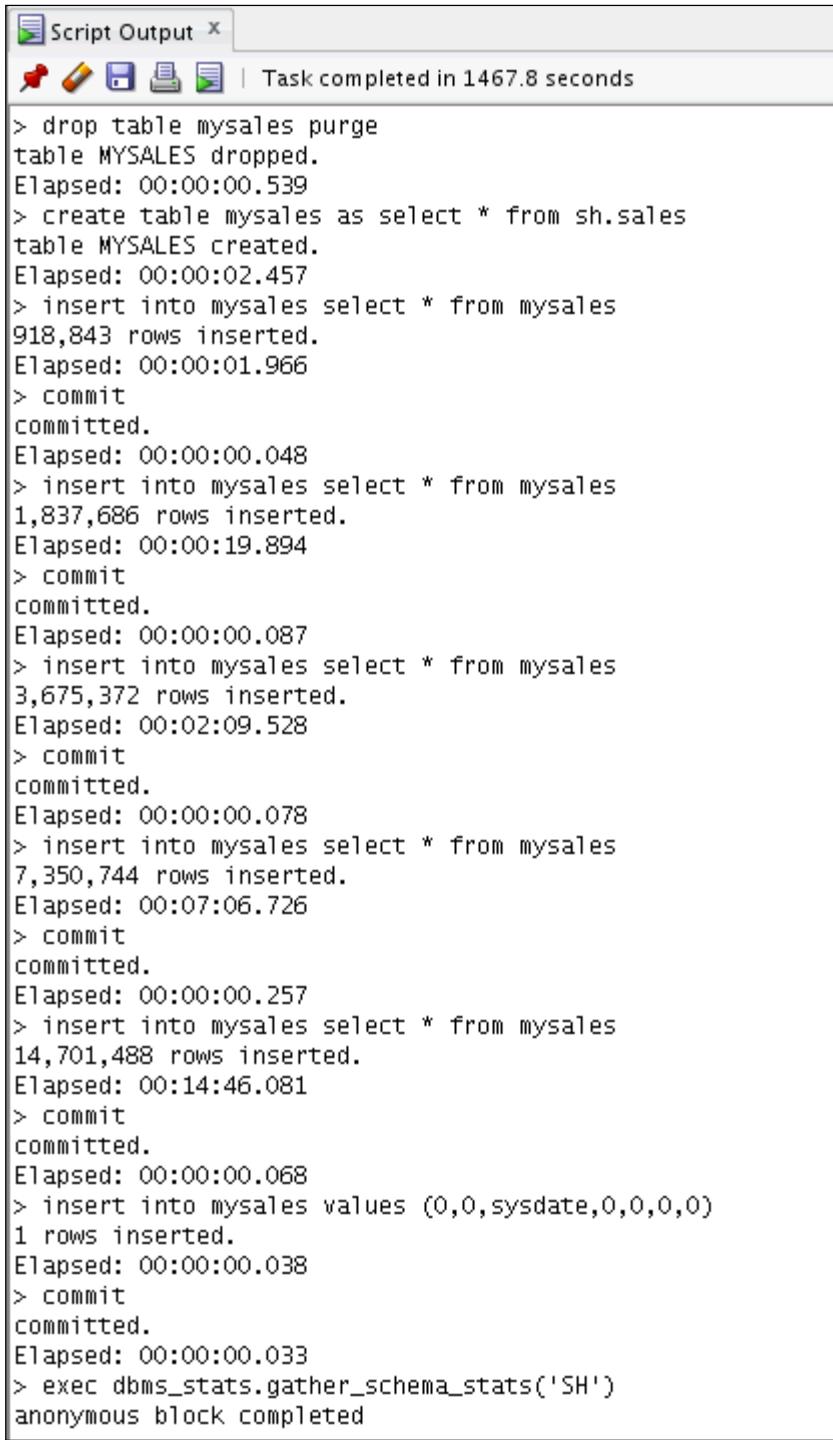
create table mysales as select * from sh.sales;

insert into mysales select * from mysales;
commit;

insert into mysales values (0,0,sysdate,0,0,0);
commit;

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

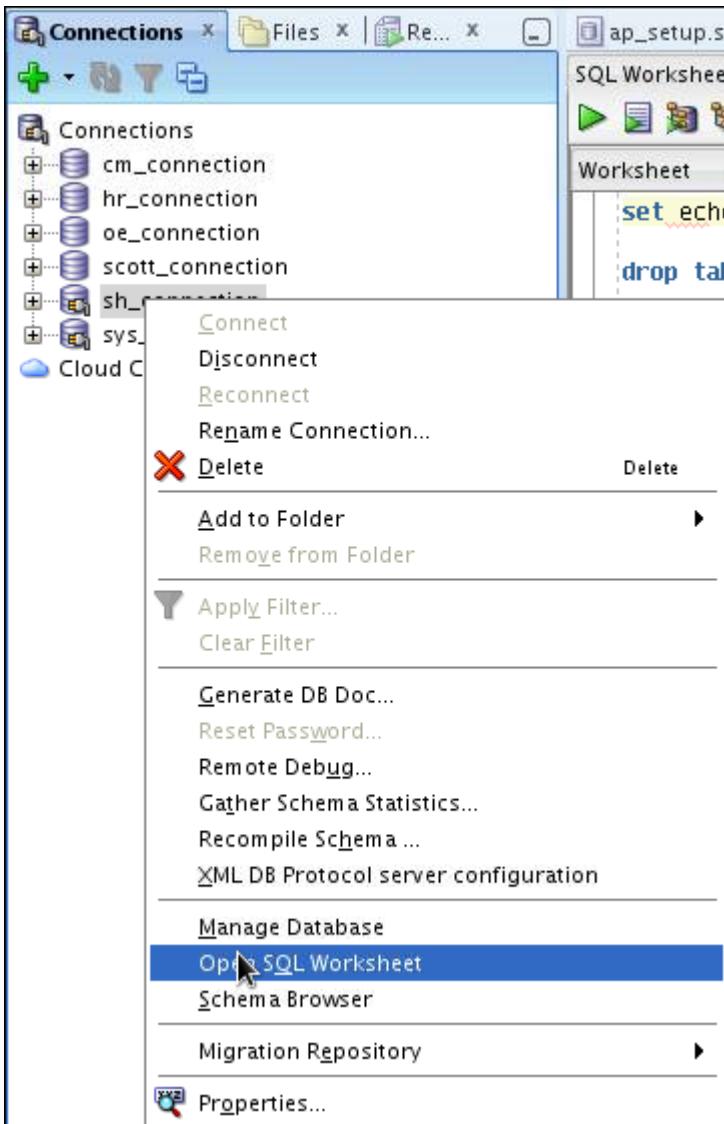
The total execution time is listed as 279.06100464 seconds.



The screenshot shows the 'Script Output' window of Oracle SQL Developer. The title bar says 'Script Output X'. Below it are icons for copy, paste, and other functions. The main area displays the following SQL script execution log:

```
> drop table mysales purge
table MYSALES dropped.
Elapsed: 00:00:00.539
> create table mysales as select * from sh.sales
table MYSALES created.
Elapsed: 00:00:02.457
> insert into mysales select * from mysales
918,843 rows inserted.
Elapsed: 00:00:01.966
> commit
committed.
Elapsed: 00:00:00.048
> insert into mysales select * from mysales
1,837,686 rows inserted.
Elapsed: 00:00:19.894
> commit
committed.
Elapsed: 00:00:00.087
> insert into mysales select * from mysales
3,675,372 rows inserted.
Elapsed: 00:02:09.528
> commit
committed.
Elapsed: 00:00:00.078
> insert into mysales select * from mysales
7,350,744 rows inserted.
Elapsed: 00:07:06.726
> commit
committed.
Elapsed: 00:00:00.257
> insert into mysales select * from mysales
14,701,488 rows inserted.
Elapsed: 00:14:46.081
> commit
committed.
Elapsed: 00:00:00.068
> insert into mysales values (0,0,sysdate,0,0,0,0)
1 rows inserted.
Elapsed: 00:00:00.038
> commit
committed.
Elapsed: 00:00:00.033
> exec dbms_stats.gather_schema_stats('SH')
anonymous block completed
```

3. What do you observe when you execute the following query in the `sh_connection` worksheet?
 - a. Right-click `sh_connection` and select Open SQL Worksheet.



- b. Autotrace the query.

```
select * from mysales where prod_id=0;
```

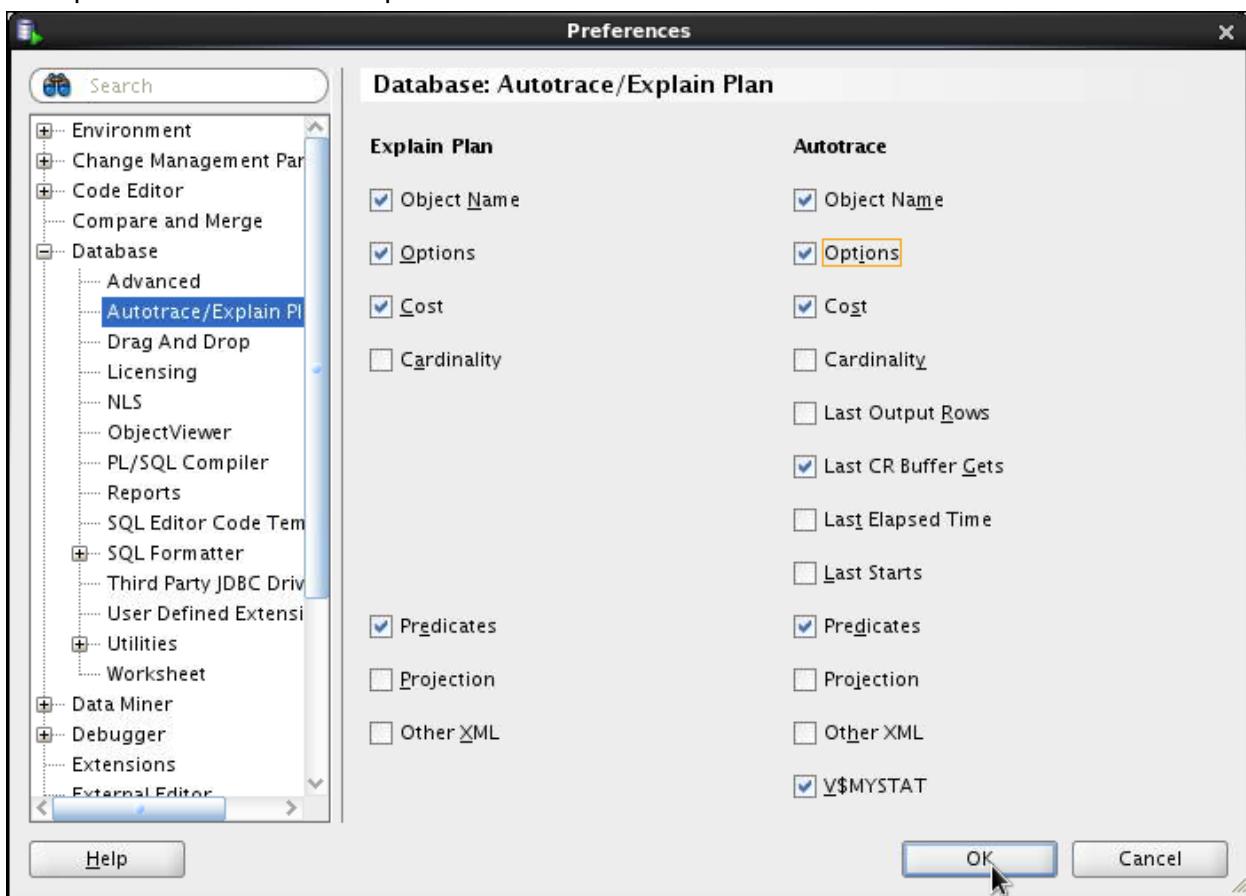


- c. Observe the output.

The screenshot shows the Oracle SQL Developer interface. At the top, there are tabs for 'ap_setup.sql', 'idx_setup.sql', and 'sh connection~1.x'. The status bar at the bottom indicates '9.2100004 seconds'. Below the tabs, there's a toolbar with various icons. The main area has two panes: 'Worksheet' and 'Query Builder'. The 'Worksheet' pane contains the SQL query: 'select * from mysales where prod_id=0;'. The 'Autotrace' pane shows the execution plan for this query. The plan details a 'SELECT STATEMENT' with an 'OBJECT_NAME' of 'MYSALES'. Underneath it, a 'TABLE ACCESS' operation is shown with 'Filter Predicates' including 'PROD_ID=0'. The 'OPTIONS' column for the 'TABLE ACCESS' row is highlighted with a red box and contains the value 'FULL'.

- Basically, there are no indexes on the MYSALES table.
- The only possibility for the optimizer is to use a full table scan to retrieve only one row. You can see that the full table scan takes a long time.

Note: To see the options as a separate column in the execution plan, go to **Tools->Preferences->Database->Autotrace/Explain Plan** and select the **Options** check box under **Autotrace**. If you do not select the Options check box, the options will be attached to the Operations column in the plan.



4. To enhance the performance of the query in step 3, re-execute the query in step 4 after executing `create_mysales_index.sql`.
- Open the `create_mysales_index.sql` file and execute it with `sh_connection`.

```
ap_setup.sql x | idx_setup.sql x | sh_connection x | create_mysales_index.sql x
SQL Worksheet History
Worksheet Query Builder
set echo on

create index mysales_prodid_idx on mysales(prod_id)
nologging compute statistics;

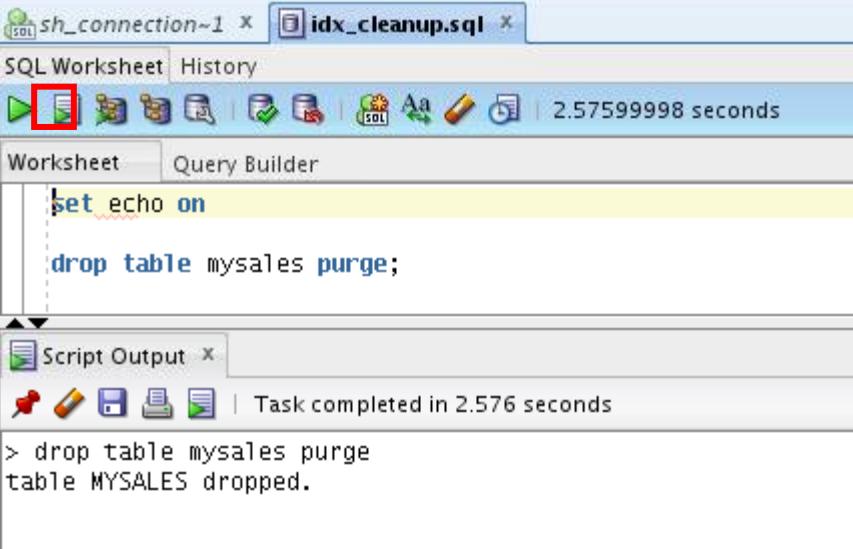
Script Output x
Task completed in 105.871 seconds
> create index mysales_prodid_idx on mysales(prod_id)
nologging compute statistics
index MYSALES_PRODID_IDX created.
```

- Autotrace the query in step 3(b) again. Observe the output.

| V\$STATNAME | Name | V\$MYSTAT | Value |
|---------------------------------|------|-----------|-------|
| Batched IO (bound) vector count | | 0 | |
| Batched IO (full) vector count | | 0 | |
| Batched IO (space) vector count | | 0 | |
| Batched IO block miss count | | 0 | |
| Batched IO buffer defrag count | | 0 | |
| Batched IO double miss count | | 0 | |

- You can see a dramatic improvement in performance. Notice the difference in time, cost, and physical reads.

5. Clean up your environment for case 1 by executing the `idx_cleanup.sql` script.

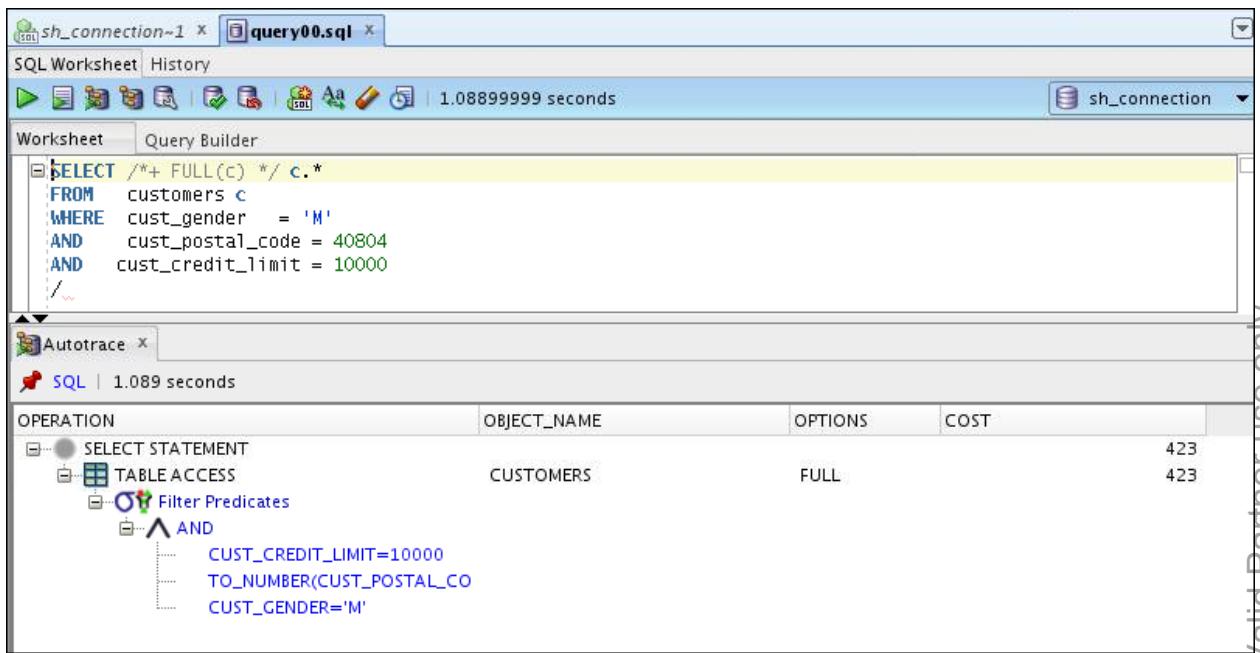


The screenshot shows the Oracle SQL Worksheet interface. A script named `idx_cleanup.sql` is open. In the worksheet pane, the command `drop table mysales purge;` is entered. The execute button (a green triangle icon) is highlighted with a red box. In the script output pane, the command is executed, and the message "table MYSALES dropped." is displayed.

6. **Case 2: Compare Single Column Index Access:** Open a terminal window. Connect to `sh` and drop all the indexes that are currently created on the `CUSTOMERS` table, except its primary key index, by executing `drop_customers_indexes.sql`.

```
$ cd /home/oracle/labs/solutions/Access_Paths/  
$ sqlplus sh/sh  
  
...  
Connected to:  
  
SQL> @drop_customers_indexes.sql
```

7. Autotrace the query in `query00.sql`. What do you observe?



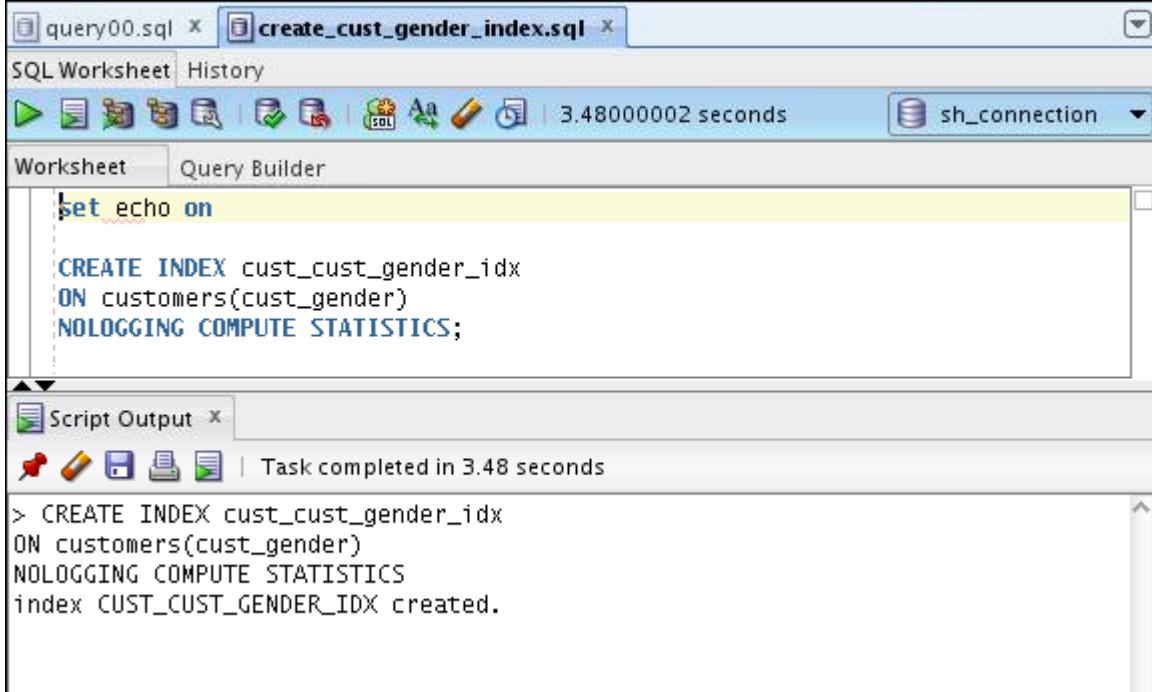
8. Create three B*-tree indexes on the following CUSTOMERS table columns by using sh_connection:

cust_gender

cust_postal_code

cust_credit_limit

- a. Open and execute the `create_cust_gender_index.sql` script.



- b. Open and execute the `create_cust_postal_code_index.sql` script.

The screenshot shows the Oracle SQL Worksheet interface. The title bar has three tabs: 'query00.sql', 'create_cust_gender_index.sql', and 'create_cust_postal_code_index.sql' (which is currently selected). The toolbar includes icons for running, saving, and zooming. The status bar shows '3.13599992 seconds'. A connection dropdown shows 'sh_connection'. The 'Worksheet' tab is active, displaying the SQL command:

```
set echo on

CREATE INDEX cust_cust_postal_code_idx
ON customers(cust_postal_code)
NOLOGGING COMPUTE STATISTICS;
```

The 'Script Output' pane below shows the execution results:

```
> CREATE INDEX cust_cust_postal_code_idx
ON customers(cust_postal_code)
NOLOGGING COMPUTE STATISTICS
index CUST_CUST_POSTAL_CODE_IDX created.
```

- c. Open and execute the `create_cust_credit_limit_index.sql` script.

The screenshot shows the Oracle SQL Worksheet interface. The title bar has three tabs: 'query00.sql', 'create_cust_credit_limit_index.sql' (selected), and 'create_cust_postal_code_index.sql'. The toolbar and status bar are similar to the previous screenshot. The 'Worksheet' tab is active, displaying the SQL command:

```
set echo on

CREATE INDEX cust_cust_credit_limit_idx
ON customers(cust_credit_limit)
NOLOGGING COMPUTE STATISTICS;
```

The 'Script Output' pane shows the execution results:

```
> CREATE INDEX cust_cust_credit_limit_idx
ON customers(cust_credit_limit)
NOLOGGING COMPUTE STATISTICS
index CUST_CUST_CREDIT_LIMIT_IDX created.
```

- d. To verify that the indexes exist, execute `list_customers_indexes.sql`.

The screenshot shows the Oracle SQL Worksheet interface. The top bar has tabs for 'query00.sql' and 'list_customers_indexes.sql'. The 'list_customers_indexes.sql' tab is active. The toolbar includes icons for file operations, search, and help. The status bar shows '2.5739998ts'. A connection dropdown shows 'sh_connection'. The main area has two panes: 'Worksheet' and 'Query Builder'. The 'Worksheet' pane contains the following SQL code:

```
SELECT ui.table_name,
       decode(ui.index_type,
              'NORMAL', ui.uniqueness
              ,ui.index_type) AS index_type
      ,ui.index_name
   FROM user_indexes ui
 WHERE ui.table_name = 'CUSTOMERS'
 ORDER BY ui.table_name
      , ui.uniqueness desc;
```

The 'Script Output' pane below shows the results of the query:

| TABLE_NAME | INDEX_TYPE | INDEX_NAME |
|------------|------------|----------------------------|
| CUSTOMERS | UNIQUE | CUSTOMERS_PK |
| CUSTOMERS | NONUNIQUE | CUST_CUST_POSTAL_CODE_IDX |
| CUSTOMERS | NONUNIQUE | CUST_CUST_CREDIT_LIMIT_IDX |
| CUSTOMERS | NONUNIQUE | CUST_CUST_GENDER_IDX |

A message at the bottom of the output pane says 'Task completed in 2.574 seconds'.

9. Start monitoring all the CUSTOMERS indexes. Notice that the value in the USED column is NO.
 - a. Open and execute the `start_monitoring_indexes.sql` script by using `sh_connection`.

```

query00.sql x start_monitoring_indexes.sql x
SQL Worksheet History
Worksheet Query Builder
set echo on

ALTER INDEX CUSTOMERS_PK MONITORING USAGE;

ALTER INDEX CUST_CUST_POSTAL_CODE_IDX MONITORING USAGE;

ALTER INDEX CUST_CUST_GENDER_IDX MONITORING USAGE;

ALTER INDEX CUST_CUST_CREDIT_LIMIT_IDX MONITORING USAGE;

Script Output x
Task completed in 0.188 seconds
> ALTER INDEX CUSTOMERS_PK MONITORING USAGE
index CUSTOMERS_PK altered.
> ALTER INDEX CUST_CUST_POSTAL_CODE_IDX MONITORING USAGE
index CUST_CUST_POSTAL_CODE_IDX altered.
> ALTER INDEX CUST_CUST_GENDER_IDX MONITORING USAGE
index CUST_CUST_GENDER_IDX altered.
> ALTER INDEX CUST_CUST_CREDIT_LIMIT_IDX MONITORING USAGE
index CUST_CUST_CREDIT_LIMIT_IDX altered.

```

- b. Open and execute the statement (Ctrl + Enter) in the show_index_usage.sql script.

```

query00.sql x show_index_usage.sql x
SQL Worksheet History
Worksheet Query Builder
select * from v$object_usage;

Query Result x
SQL | All Rows Fetched: 4 in 0.053 seconds
INDEX_NAME TABLE_NAME MONITORING USED START_MONITORING END_MONITORING
1 CUSTOMERS_PK CUSTOMERS YES NO 03/27/2013 23:44:11 (null)
2 CUST_CUST_POSTAL_CODE_IDX CUSTOMERS YES NO 03/27/2013 23:44:11 (null)
3 CUST_CUST_GENDER_IDX CUSTOMERS YES NO 03/27/2013 23:44:11 (null)
4 CUST_CUST_CREDIT_LIMIT_IDX CUSTOMERS YES NO 03/27/2013 23:44:11 (null)

```

10. Autotrace the query in query01.sql. What do you observe?

```

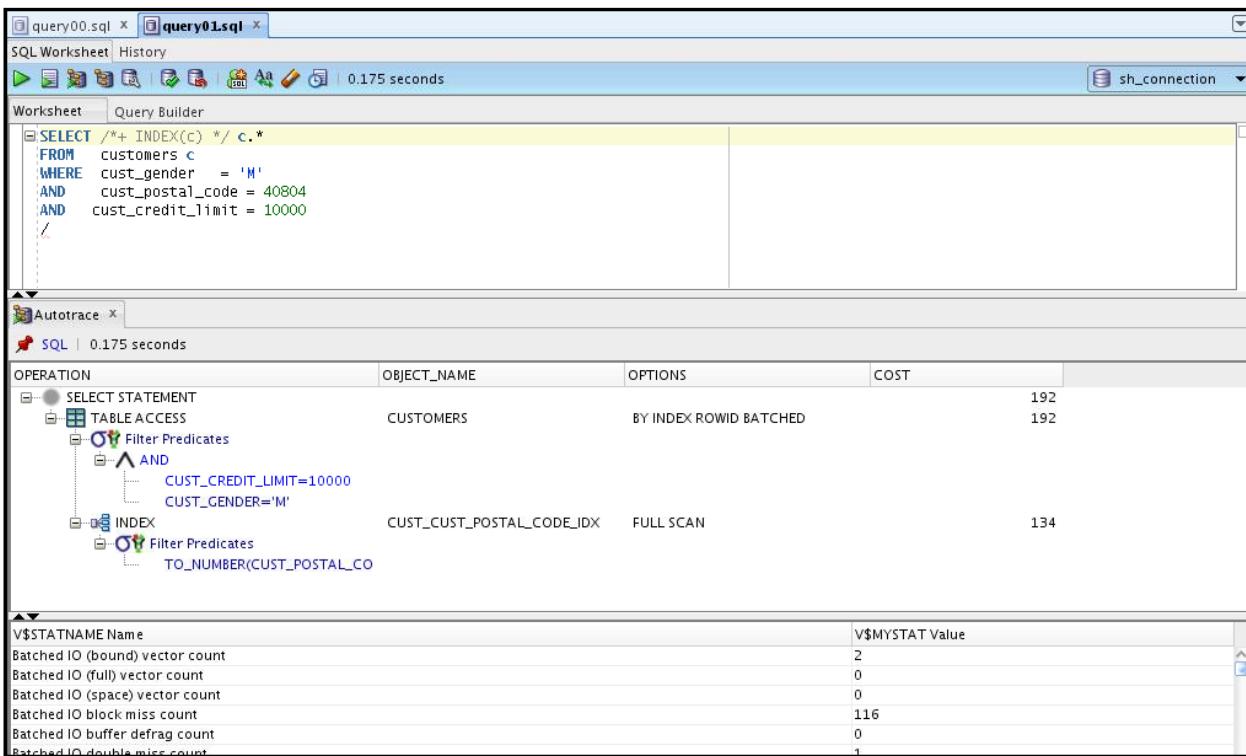
SELECT /*+ INDEX(c) */ c.*
FROM   customers c
WHERE  cust_gender    = 'M'
AND    cust_postal_code = 40804

```

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

```
AND    cust_credit_limit = 10000
/
```

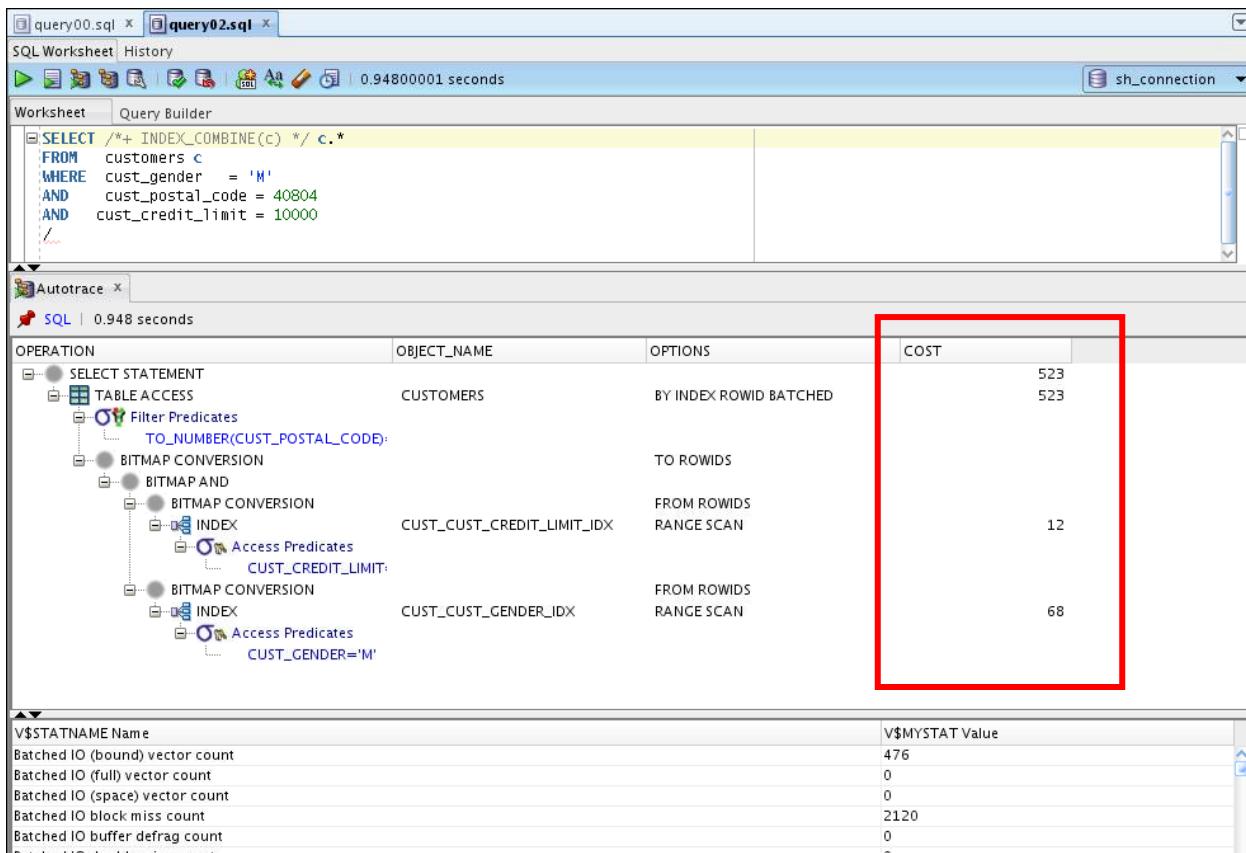
Hint: Check the estimated cost in the execution plan and the sum of db block gets and consistent gets in the statistics.



- The optimizer chooses to use only one index to do a full scan. The cost is lower than the full table scan.

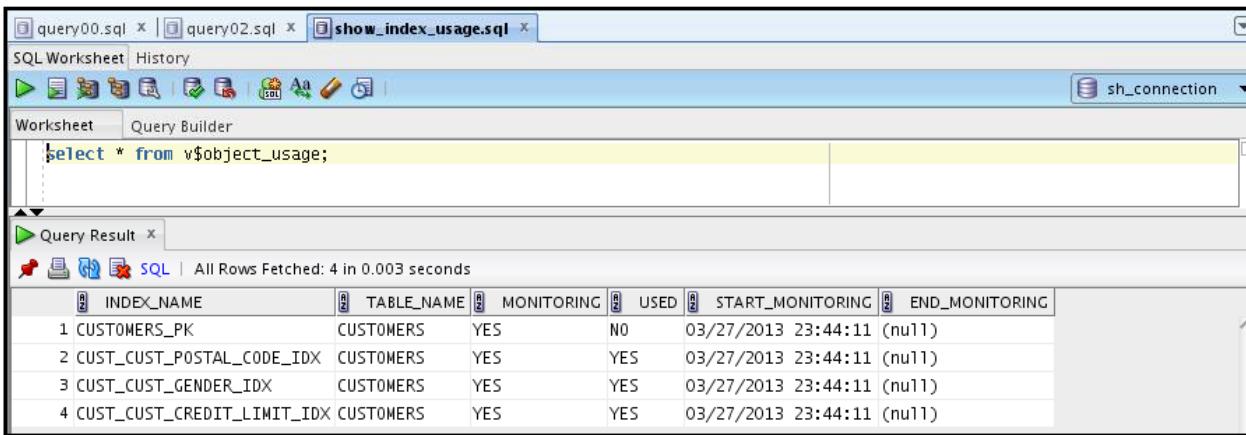
11. Autotrace the query in query02.sql by using sh_connection. What do you observe?

```
SELECT /*+ INDEX_COMBINE(c) */ c.*  
FROM   customers c  
WHERE  cust_gender  = 'M'  
AND    cust_postal_code = 40804  
AND    cust_credit_limit = 10000;
```



- This time the optimizer uses multiple indexes and combines them to access the table. However, the cost is higher than that from the previous step, but is still lower than the full table scan.

12. Execute `show_index_usage.sql` to confirm the list of indexes that were accessed in this case.

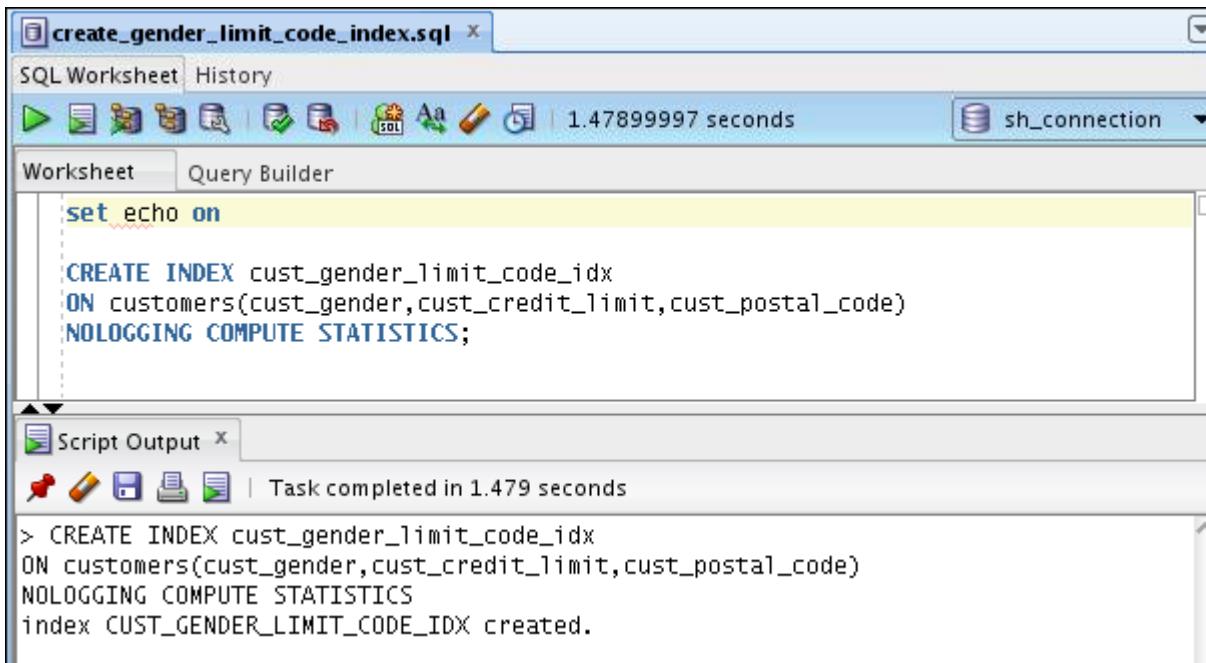


13. **Case 3: Concatenated Index:** Open a terminal window. Connect to `sh` and drop all the indexes that are currently created on the `CUSTOMERS` table, except its primary key index, by executing `drop_customers_indexes.sql`.

```
$ cd /home/oracle/labs/solutions/Access_Paths/  
$ sqlplus sh/sh  
  
...  
Connected to:  
  
SQL> @drop_customers_indexes.sql
```

14. Open and execute the `create_gender_limit_code_index.sql` script by using `sh_connection` to make sure that you create a concatenated index on the following CUSTOMERS columns, and in the order mentioned here:

`cust_gender`
`cust_credit_limit`
`cust_postal_code`



15. Autotrace the query in `query01.sql`. What do you observe?

Autotrace output for query01.sql:

```

SELECT /*+ INDEX(c) */ c.*
  FROM customers c
 WHERE cust_gender = 'M'
   AND cust_postal_code = 40804
   AND cust_credit_limit = 10000
  /

```

Execution Plan:

| OPERATION | OBJECT_NAME | OPTIONS | COST |
|--------------------------|----------------------------|------------------------|------|
| SELECT STATEMENT | | | 19 |
| TABLE ACCESS | CUSTOMERS | BY INDEX ROWID BATCHED | 19 |
| INDEX | CUST_GENDER_LIMIT_CODE_IDX | RANGE SCAN | 14 |
| Access Predicates | | | |
| AND | | | |
| CUST_GENDER='M' | | | |
| CUST_CREDIT_LIMIT=10000 | | | |
| Filter Predicates | | | |
| TO_NUMBER(CUST_POSTAL_CO | | | |

V\$STATNAME and V\$MYSTAT:

| V\$STATNAME Name | V\$MYSTAT Value |
|---------------------------------|-----------------|
| Batched IO (bound) vector count | 0 |
| Batched IO (full) vector count | 0 |
| Batched IO (space) vector count | 0 |
| Batched IO block miss count | 4 |
| Batched IO buffer defrag count | 0 |

- The optimizer uses your concatenated index and the resulting cost is by far the best compared to the previous steps.

16. Autotrace the query in query03.sql. What do you observe?

Autotrace output for query03.sql:

```

SELECT /*+ INDEX(c) */ c.*
  FROM customers c
 WHERE cust_gender = 'M'
   AND cust_credit_limit = 10000
  /

```

Execution Plan:

| OPERATION | OBJECT_NAME | OPTIONS | COST |
|-------------------------|----------------------------|------------------------|------|
| SELECT STATEMENT | | | 3961 |
| TABLE ACCESS | CUSTOMERS | BY INDEX ROWID BATCHED | 3961 |
| INDEX | CUST_GENDER_LIMIT_CODE_IDX | RANGE SCAN | 14 |
| Access Predicates | | | |
| AND | | | |
| CUST_GENDER='M' | | | |
| CUST_CREDIT_LIMIT=10000 | | | |

V\$STATNAME and V\$MYSTAT:

| V\$STATNAME Name | V\$MYSTAT Value |
|---------------------------------|-----------------|
| Batched IO (bound) vector count | 1 |
| Batched IO (full) vector count | 0 |
| Batched IO (space) vector count | 0 |
| Batched IO block miss count | 46 |

- The query is almost the same as in the previous step, but the predicate on cust_postal_code is removed. The optimizer can still use the concatenated index, but the resulting cost is much higher because neither cust_credit_limit nor cust_gender are very selective.

17. Autotrace the query in query04.sql. What do you observe?

Autotrace report for query04.sql:

| OPERATION | OBJECT_NAME | OPTIONS | COST |
|-------------------|----------------------------|------------------------|------|
| SELECT STATEMENT | | | 157 |
| TABLE ACCESS | CUSTOMERS | BY INDEX ROWID BATCHED | 157 |
| INDEX | CUST_GENDER_LIMIT_CODE_IDX | RANGE SCAN | 116 |
| Access Predicates | CUST_GENDER='M' | | |
| Filter Predicates | TO_NUMBER(CUST_POSTAL_CO | | |

| V\$STATNAME Name | V\$MYSTAT Value |
|---------------------------------|-----------------|
| Batched IO (bound) vector count | 1 |
| Batched IO (full) vector count | 0 |
| Batched IO (space) vector count | 0 |
| Batched IO block miss count | 47 |
| Batched IO buffer defrag count | 0 |

- You replaced `cust_credit_limit` with `cust_postal_code`, which has better selectivity. The index is used and the resulting cost is better.

18. Autotrace the query in `query05.sql`. What do you observe?

Autotrace report for query05.sql:

| OPERATION | OBJECT_NAME | OPTIONS | COST |
|-------------------------|----------------------------|------------------------|------|
| SELECT STATEMENT | | | 179 |
| TABLE ACCESS | CUSTOMERS | BY INDEX ROWID BATCHED | 179 |
| INDEX | CUST_GENDER_LIMIT_CODE_IDX | SKIP SCAN | 172 |
| Access Predicates | CUST_CREDIT_LIMIT=10000 | | |
| Filter Predicates | AND | | |
| CUST_CREDIT_LIMIT=10000 | | | |
| TO_NUMBER(CUST_POSTAL | | | |

| V\$STATNAME Name | V\$MYSTAT Value |
|---------------------------------|-----------------|
| Batched IO (bound) vector count | 0 |
| Batched IO (full) vector count | 0 |
| Batched IO (space) vector count | 0 |
| Batched IO block miss count | 13 |

- The leading part of the concatenated index is no longer part of the query. However, the optimizer is still able to use the index by doing an index skip scan.

19. Case 4: Bitmap Index Access: Open a terminal window. Connect to `sh` and drop all the indexes that are currently created on the CUSTOMERS table, except its primary key index, by executing `drop_customers_indexes.sql`.

```
$ cd /home/oracle/labs/solutions/Access_Paths/
$ sqlplus sh/sh
```

```

...
Connected to:...

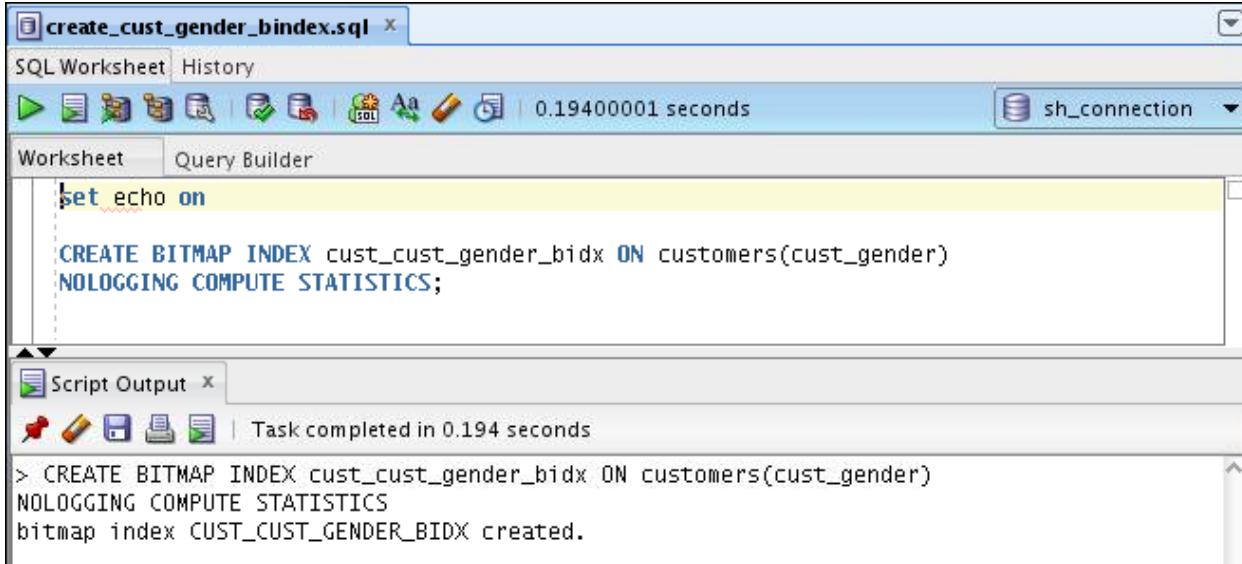
SQL> @drop_customers_indexes.sql

```

20. Open and execute three scripts by using sh_connection to create three different bitmap indexes on the following columns of the CUSTOMERS table:

cust_gender (create_cust_gender_bindex.sql)
 cust_postal_code (create_cust_postal_code_bindex.sql)
 cust_credit_limit (create_cust_credit_limit_bindex.sql)

- a. Open and execute the create_cust_gender_bindex.sql script.



The screenshot shows the Oracle SQL Developer interface. The title bar says "create_cust_gender_bindex.sql". The toolbar includes icons for running, saving, and zooming. The connection dropdown says "sh_connection". The main workspace has tabs for "Worksheet" and "Query Builder", with "Worksheet" selected. The code in the workspace is:

```

set echo on

CREATE BITMAP INDEX cust_cust_gender_bidx ON customers(cust_gender)
NOLOGGING COMPUTE STATISTICS;

```

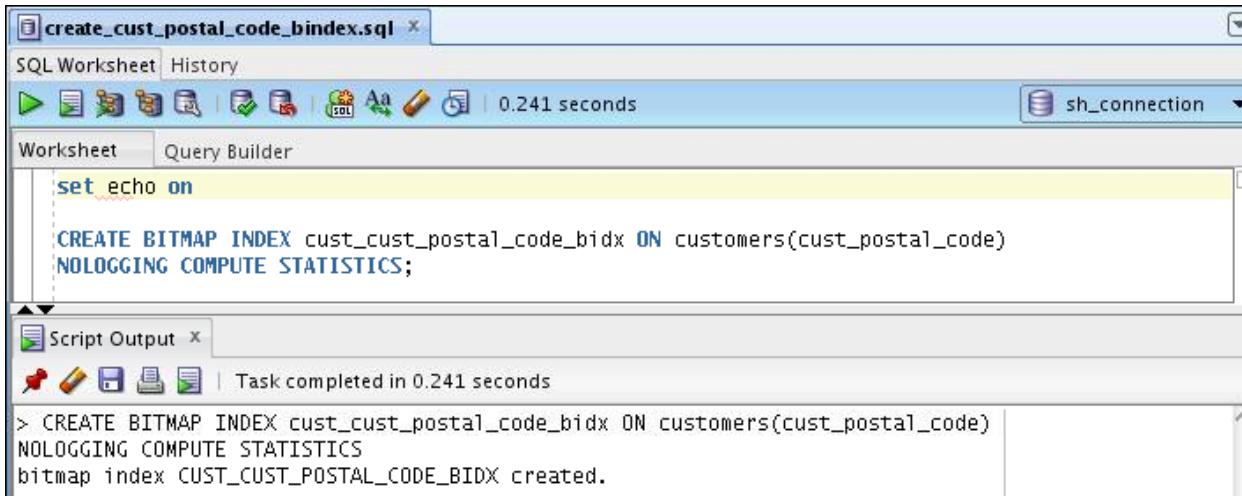
The "Script Output" window below shows the execution results:

```

Task completed in 0.194 seconds
> CREATE BITMAP INDEX cust_cust_gender_bidx ON customers(cust_gender)
NOLOGGING COMPUTE STATISTICS
bitmap index CUST_CUST_GENDER_BIDX created.

```

- b. Open and execute the create_cust_postal_code_bindex.sql script.



The screenshot shows the Oracle SQL Developer interface. The title bar says "create_cust_postal_code_bindex.sql". The toolbar includes icons for running, saving, and zooming. The connection dropdown says "sh_connection". The main workspace has tabs for "Worksheet" and "Query Builder", with "Worksheet" selected. The code in the workspace is:

```

set echo on

CREATE BITMAP INDEX cust_cust_postal_code_bidx ON customers(cust_postal_code)
NOLOGGING COMPUTE STATISTICS;

```

The "Script Output" window below shows the execution results:

```

Task completed in 0.241 seconds
> CREATE BITMAP INDEX cust_cust_postal_code_bidx ON customers(cust_postal_code)
NOLOGGING COMPUTE STATISTICS
bitmap index CUST_CUST_POSTAL_CODE_BIDX created.

```

- c. Open and execute the create_cust_credit_limit_bindex.sql script.

The screenshot shows the Oracle SQL Developer interface. The top tab bar has two tabs: "create_cust_credit_limit_bindex.sql" and "list_customers_indexes.sql". The "create_cust_credit_limit_bindex.sql" tab is active, showing a SQL Worksheet with the following code:

```
set echo on
CREATE BITMAP INDEX cust_cust_credit_limit_bidx ON customers(cust_credit_limit)
NOLOGGING COMPUTE STATISTICS;
```

Below the worksheet is a "Script Output" window showing the execution results:

```
> CREATE BITMAP INDEX cust_cust_credit_limit_bidx ON customers(cust_credit_limit)
NOLOGGING COMPUTE STATISTICS
bitmap index CUST_CUST_CREDIT_LIMIT_BIDX created.
```

- d. Confirm that the indexes have been created by executing the statement in the `list_customers_indexes.sql` script.

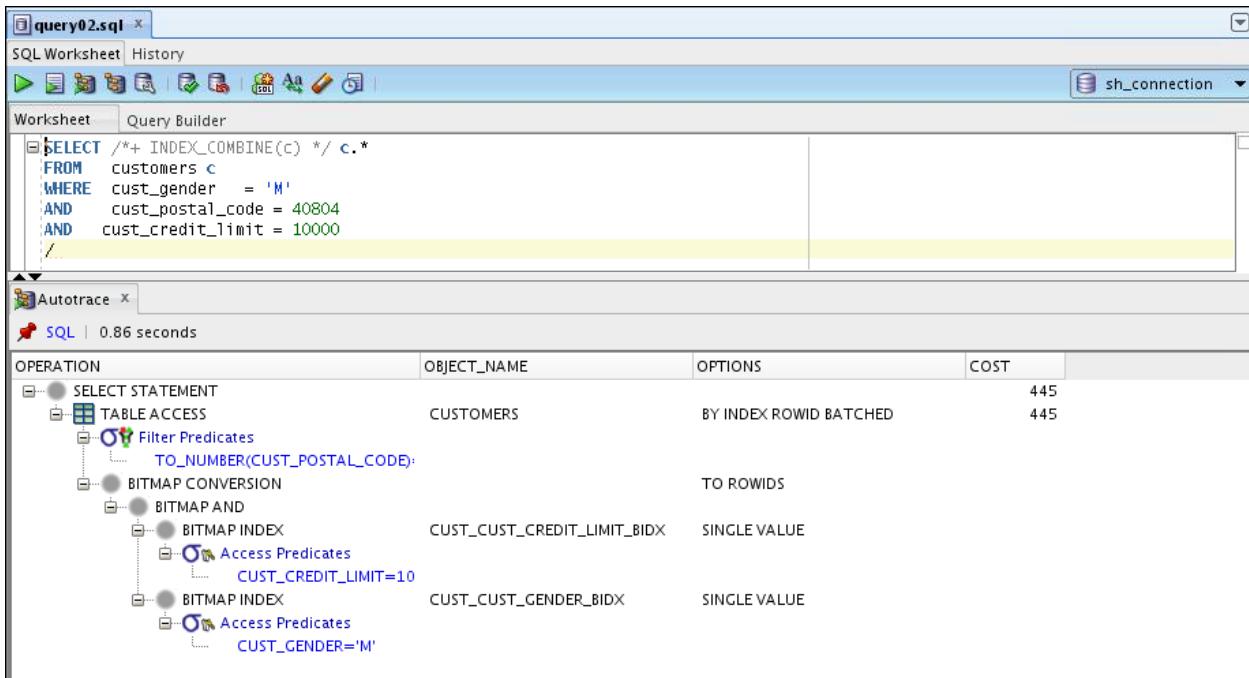
The screenshot shows the Oracle SQL Developer interface with the "list_customers_indexes.sql" tab active. A green box highlights the play button icon in the toolbar, indicating the script is ready to be executed. The SQL Worksheet contains the following query:

```
SELECT ui.table_name
      , decode(ui.index_type
              , 'NORMAL' , ui.uniqueness
              , ui.index_type) AS index_type
      , ui.index_name
   FROM user_indexes ui
 WHERE ui.table_name = 'CUSTOMERS'
 ORDER BY ui.table_name
      , ui.uniqueness desc;
```

Below the worksheet is a "Query Result" window showing the output of the query:

| TABLE_NAME | INDEX_TYPE | INDEX_NAME |
|-------------|------------|-----------------------------|
| 1 CUSTOMERS | UNIQUE | CUSTOMERS_PK |
| 2 CUSTOMERS | BITMAP | CUST_CUST_POSTAL_CODE_BIDX |
| 3 CUSTOMERS | BITMAP | CUST_CUST_CREDIT_LIMIT_BIDX |
| 4 CUSTOMERS | BITMAP | CUST_CUST_GENDER_BIDX |

21. Autotrace the query in `query02.sql`. What do you observe?



- The optimizer uses only two bitmap indexes to solve this query. However, the cost is not good. The cost is a little lower than the cost of the full table scan.

22. **Case 5: Complex Predicate with Bitmap Indexes:** Open a terminal window. Connect to sh and drop all the indexes that are currently created on the CUSTOMERS table, except its primary key index, by executing drop_customers_indexes.sql.

```

$ cd /home/oracle/labs/solutions/Access_Paths/
$ sqlplus sh/sh

...
Connected to:...

SQL> @drop_customers_indexes.sql

```

23. After this, create two bitmap indexes on the following columns of the CUSTOMERS table:

`cust_year_of_birth` (`create_cust_year_of_birth_bindex.sql`)

`cust_credit_limit` (`create_cust_credit_limit_bindex.sql`)

- a. Open and execute the `create_cust_year_of_birth_bindex.sql` script.

The screenshot shows the Oracle SQL Worksheet interface. The title bar says "create_cust_year_of_birth_bindex.sql". The worksheet tab is selected. The code entered is:

```
set echo on

CREATE BITMAP INDEX cust_cust_year_of_birth_bidx
ON customers(cust_year_of_birth)
NOLANDING COMPUTE STATISTICS;
```

The script output window below shows the command executed and its result:

```
> CREATE BITMAP INDEX cust_cust_year_of_birth_bidx
ON customers(cust_year_of_birth)
NOLANDING COMPUTE STATISTICS
bitmap index CUST_CUST_YEAR_OF_BIRTH_BIDX created.
```

- b. Open and execute the `create_cust_credit_limit_bindex.sql` script.

The screenshot shows the Oracle SQL Worksheet interface. The title bar says "create_cust_credit_limit_bindex.sql". The worksheet tab is selected. The code entered is:

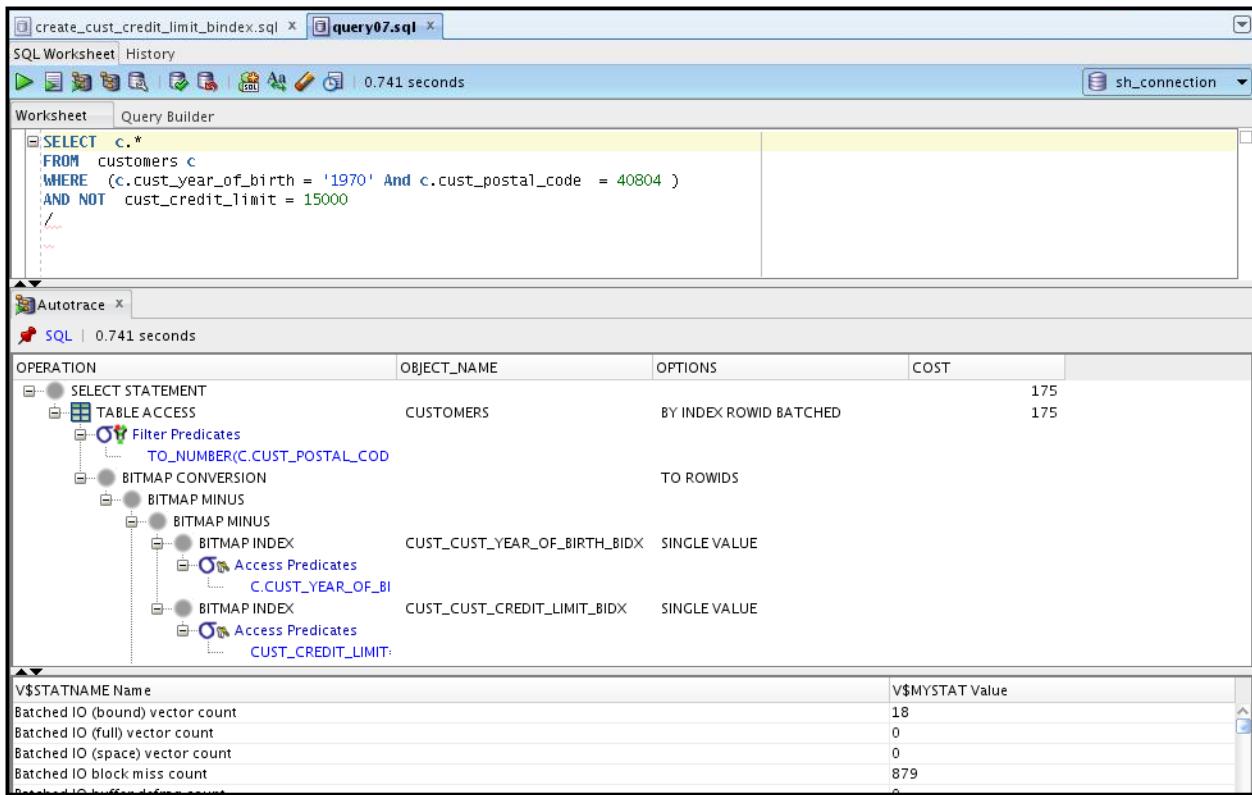
```
set echo on

CREATE BITMAP INDEX cust_cust_credit_limit_bidx ON customers(cust_credit_limit)
NOLANDING COMPUTE STATISTICS;
```

The script output window below shows the command executed and its result:

```
> CREATE BITMAP INDEX cust_cust_credit_limit_bidx ON customers(cust_credit_limit)
NOLANDING COMPUTE STATISTICS
bitmap index CUST_CUST_CREDIT_LIMIT_BIDX created.
```

24. Autotrace the query in `query07.sql`. What do you observe?



- The query has a complex WHERE clause that is well suited for using bitmap indexes. The optimizer uses two bitmap indexes and the resulting cost is better than the full table scan cost.

25. Make sure that the optimizer can no longer use the bitmap index that you created on the cust_year_of_birth column.

The best solution is to render it invisible. Open and execute the alter_cust_yob_idx.sql script. Verify that the optimizer_use_invisible_indexes parameter is set to FALSE.

```

select * from v$parameter
where name = 'optimizer_use_invisible_indexes';

alter index cust_cust_year_of_birth_bidx invisible;

select index_name, visibility from user_indexes
where table_owner='SH' and table_name='CUSTOMERS';

```

```

select * from v$parameter
where name = 'optimizer_use_invisible_indexes';

alter index cust_cust_year_of_birth_bidx invisible;

select index_name, visibility from user_indexes
where table_owner='SH' and table_name='CUSTOMERS';

```

| NUM | NAME | TYPE | VALUE |
|------|---------------------------------|------|---------|
| 2707 | optimizer_use_invisible_indexes | | 1 FALSE |

index CUST_CUST_YEAR_OF_BIRTH_BIDX altered.

| INDEX_NAME | VISIBILITY |
|------------------------------|------------|
| CUST_CUST_CREDIT_LIMIT_BIDX | VISIBLE |
| CUST_CUST_YEAR_OF_BIRTH_BIDX | INVISIBLE |
| CUSTOMERS_PK | VISIBLE |

26. Autotrace the query in `query07.sql`. What do you observe?

```

SELECT c.*
FROM customers c
WHERE (c.cust_year_of_birth = '1970' And c.cust_postal_code = 40804 )
AND NOT cust_credit_limit = 15000

```

| OPERATION | OBJECT_NAME | OPTIONS | COST |
|-------------------------------------|-------------|---------|------|
| SELECT STATEMENT | | | 423 |
| TABLE ACCESS | CUSTOMERS | FULL | 423 |
| Filter Predicates | | | |
| AND | | | |
| C.CUST_YEAR_OF_BIRTH=1970 | | | |
| TO_NUMBER(C.CUST_POSTAL_CODE)=40804 | | | |
| CUST_CREDIT_LIMIT<>15000 | | | |

| V\$STATNAME | Name | V\$MYSTAT | Value |
|---------------------------------|------|-----------|-------|
| Batched IO (bound) vector count | | 0 | |
| Batched IO (full) vector count | | 0 | |
| Batched IO (space) vector count | | 0 | |

- This is the same query as in step 25. However, the optimizer can no longer find a good plan that uses bitmap indexes.

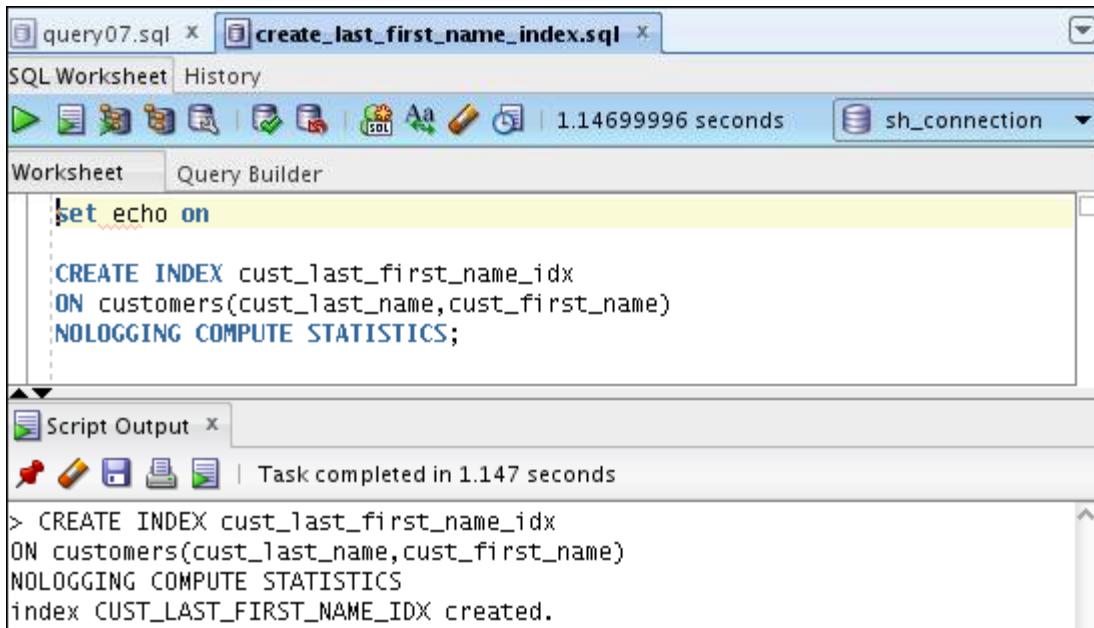
27. **Case 6: Index Only Access:** Open a terminal window. Connect to `sh` and drop all the indexes that are currently created on the `CUSTOMERS` table, except its primary key index, by executing `drop_customers_indexes.sql`.

```
$ cd /home/oracle/labs/solutions/Access_Paths/  
$ sqlplus sh/sh  
  
...  
Connected to:  
  
SQL> @drop_customers_indexes.sql
```

28. After this, use `create_last_first_name_index.sql` to create a concatenated B*-tree index on the following columns of the `CUSTOMERS` table, and in the order shown here:

`cust_last_name`
`cust_first_name`

Open and execute the `create_last_first_name_index.sql` script.



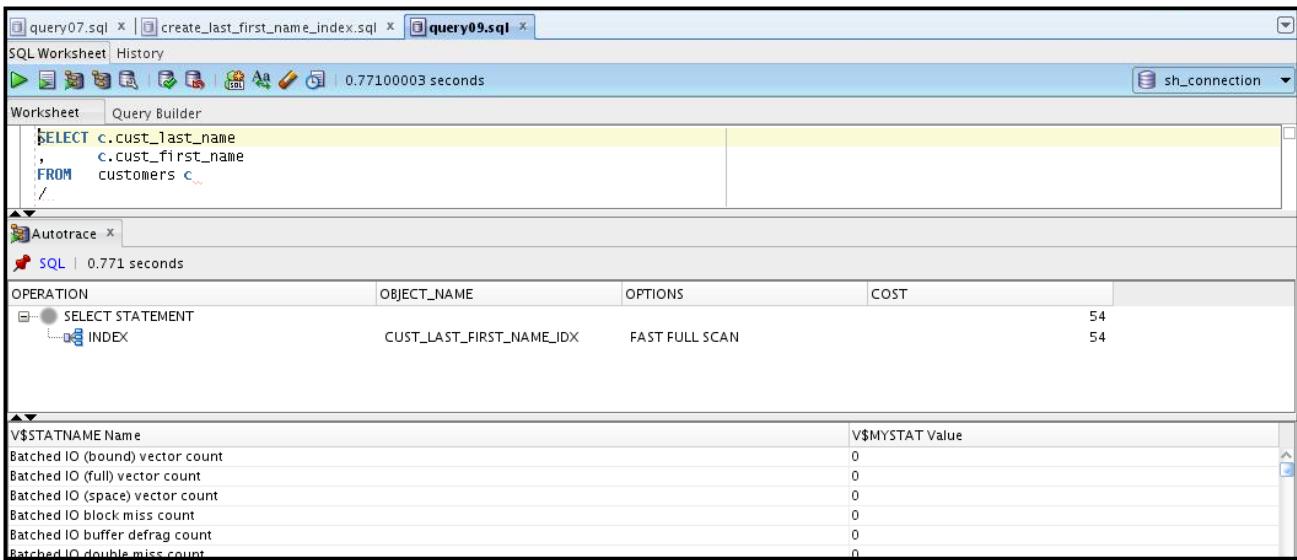
The screenshot shows the Oracle SQL Worksheet interface. The top tab bar has two tabs: "query07.sql" and "create_last_first_name_index.sql", with "create_last_first_name_index.sql" currently selected. Below the tabs is a toolbar with various icons. The main workspace is titled "Worksheet" and contains the following SQL code:

```
set echo on  
  
CREATE INDEX cust_last_first_name_idx  
ON customers(cust_last_name,cust_first_name)  
NOLOGGING COMPUTE STATISTICS;
```

Below the workspace is a "Script Output" window with the following log:

```
Task completed in 1.147 seconds  
> CREATE INDEX cust_last_first_name_idx  
ON customers(cust_last_name,cust_first_name)  
NOLOGGING COMPUTE STATISTICS  
index CUST_LAST_FIRST_NAME_IDX created.
```

29. Autotrace the query in `query09.sql`. What do you observe?



- The optimizer can use the index to retrieve the entire select list without accessing the table itself. The resulting cost is very good.
30. **Case 7: Index Join:** Open a terminal window. Connect to sh and drop all the indexes that are currently created on the CUSTOMERS table, except its primary key index, by executing drop_customers_indexes.sql.

```
$ cd /home/oracle/labs/solutions/Access_Paths/
$ sqlplus sh/sh

...
Connected to:...

SQL> @drop_customers_indexes.sql
```

31. After this, create two B*-tree indexes on the following columns of the CUSTOMERS table:
 cust_last_name (create_last_name.index.sql)
 cust_first_name (create_first_name.index.sql)
- Open and execute the create_last_name_index.sql script.

The screenshot shows the Oracle SQL Worksheet interface. The title bar says "create_last_name_index.sql". The toolbar includes icons for New, Open, Save, Print, Find, Copy, Paste, Undo, Redo, and others. The status bar at the top right shows "1.17400002 seconds" and a connection named "sh_connection". The main area is a "Worksheet" tab, containing the following SQL code:

```
set echo on

CREATE INDEX cust_cust_last_name_idx
ON customers(cust_last_name)
NOLOGGING COMPUTE STATISTICS;
```

Below the worksheet is a "Script Output" window showing the execution results:

```
> CREATE INDEX cust_cust_last_name_idx
ON customers(cust_last_name)
NOLOGGING COMPUTE STATISTICS
index CUST_CUST_LAST_NAME_IDX created.
```

- b. Open and execute the `create_first_name_index.sql` script.

The screenshot shows the Oracle SQL Worksheet interface. The title bar says "create_first_name_index.sql". The toolbar and status bar are identical to the previous screenshot. The main area is a "Worksheet" tab, containing the following SQL code:

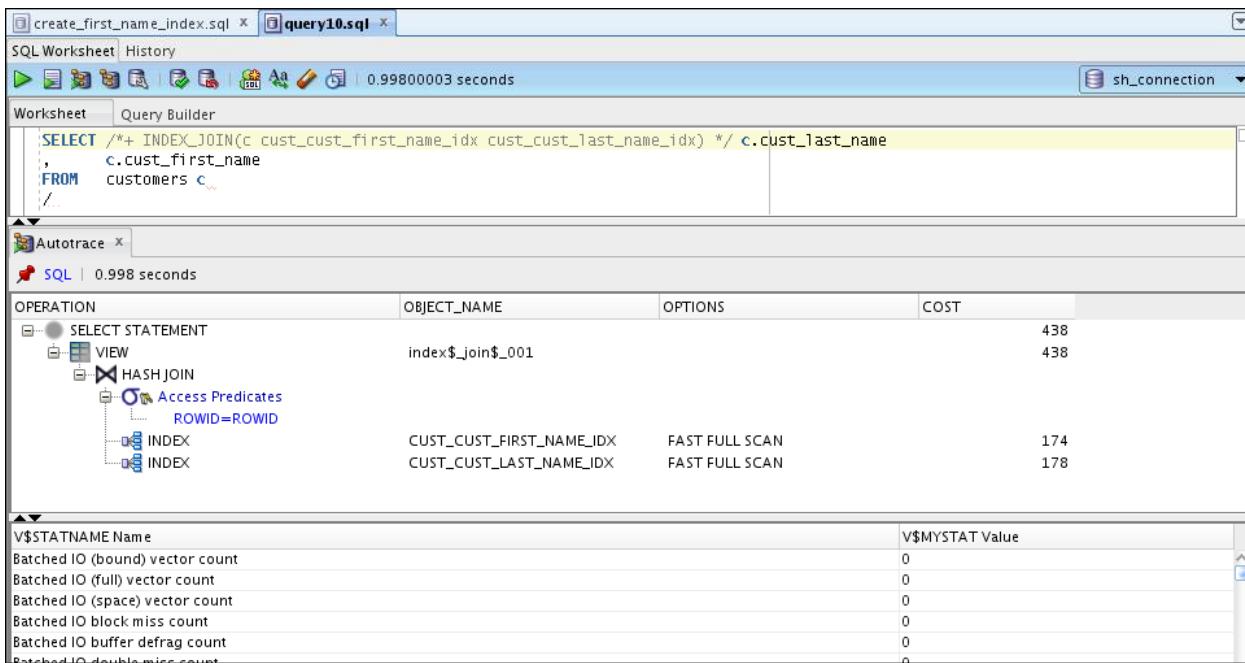
```
set echo on

CREATE INDEX cust_cust_first_name_idx
ON customers(cust_first_name)
NOLOGGING COMPUTE STATISTICS;
```

Below the worksheet is a "Script Output" window showing the execution results:

```
> CREATE INDEX cust_cust_first_name_idx
ON customers(cust_first_name)
NOLOGGING COMPUTE STATISTICS
index CUST_CUST_FIRST_NAME_IDX created.
```

32. Autotrace the query in the `query10.sql` script. What do you observe?



- Although the optimizer can use both the indexes, the resulting cost is not better than the concatenated index.

33. **Case 8: Bitmap Index Only Access:** Open a terminal window. Connect to sh and drop all the indexes that are currently created on the CUSTOMERS table, except its primary key index, by executing `drop_customers_indexes.sql`.

```
$ cd /home/oracle/labs/solutions/Access_Paths/
$ sqlplus sh/sh

...
Connected to:...

SQL> @drop_customers_indexes.sql
```

34. Then create one bitmap index on the following column of the CUSTOMERS table:
`cust_credit_limit` (`create_cust_credit_limit_bindex.sql`)
 You can open and execute the `create_cust_credit_limit_bindex.sql` script.

The screenshot shows the Oracle SQL Developer interface. In the top-left pane, titled 'create_cust_credit_limit_bindex.sql', there is a 'Worksheet' tab containing the SQL command:

```
set echo on
CREATE BITMAP INDEX cust_cust_credit_limit_bidx ON customers(cust_credit_limit)
NOLOGGING COMPUTE STATISTICS;
```

In the bottom-right pane, titled 'Script Output', the output of the command is shown:

```
> CREATE BITMAP INDEX cust_cust_credit_limit_bidx ON customers(cust_credit_limit)
NOLOGGING COMPUTE STATISTICS
bitmap index CUST_CUST_CREDIT_LIMIT_BIDX created.
```

35. Auttrace the query in `query11.sql`. What do you observe?

The screenshot shows the Oracle SQL Developer interface. In the top-left pane, titled 'query11.sql', there is a 'Worksheet' tab containing the SQL command:

```
SELECT count(*) credit_limit
FROM   customers
WHERE  cust_credit_limit = 10000
/
```

In the bottom-right pane, titled 'Auttrace', the execution plan is displayed:

| OPERATION | OBJECT_NAME | OPTIONS | COST |
|-------------------|-----------------------------|--------------|------|
| SELECT STATEMENT | | | 1 |
| SORT | | AGGREGATE | |
| BITMAP CONVERSION | | COUNT | 1 |
| BITMAP INDEX | CUST_CUST_CREDIT_LIMIT_BIDX | SINGLE VALUE | |
| Access Predicates | CUST_CREDIT_LIMIT=1000C | | |

Below the execution plan, a table of statistics is shown:

| V\$STATNAME | Name | V\$MYSTAT | Value |
|---------------------------------|------|-----------|-------|
| Batched IO (bound) vector count | | 0 | |
| Batched IO (full) vector count | | 0 | |
| Batched IO (space) vector count | | 0 | |
| Batched IO block miss count | | 0 | |

- Although `cust_credit_limit` is not a selective column, the COUNT operation on its bitmap index is very efficient.

36. **Case 9: B*-Tree Index Only Access:** Drop all the CUSTOMERS indexes, except its primary key index. Open a terminal window. Connect to sh and drop all the indexes that are currently created on the CUSTOMERS table, except its primary key index, by executing `drop_customers_indexes.sql`.

```
$ cd /home/oracle/labs/solutions/Access_Paths/
$ sqlplus sh/sh
...

```

Connected to:...

SQL> @drop_customers_indexes.sql

37. After this, create one B*-tree index on the following column of the CUSTOMERS table:

cust_credit_limit (create_cust_credit_limit_index.sql)

Open and execute the create_cust_credit_limit_index.sql script.

```
create_cust_credit_limit_index.sql
SQL Worksheet History
Worksheet Query Builder
set echo on

CREATE INDEX cust_cust_credit_limit_idx
ON customers(cust_credit_limit)
NOLOGGING COMPUTE STATISTICS;

Script Output
Task completed in 0.804 seconds
> CREATE INDEX cust_cust_credit_limit_idx
ON customers(cust_credit_limit)
NOLOGGING COMPUTE STATISTICS
index CUST_CUST_CREDIT_LIMIT_IDX created.
```

38. Autotrace the query in query11.sql. What do you observe?

```
query11.sql
SQL Worksheet History
Worksheet Query Builder
SELECT count(*) credit_limit
FROM customers
WHERE cust_credit_limit = 10000
/
Autotrace
SQL | 0.715 seconds
OPERATION          OBJECT_NAME          OPTIONS          COST
SELECT STATEMENT
  SORT
    INDEX          CUST_CUST_CREDIT_LIMIT_IDX   AGGREGATE          12
      ACCESS PREDICATES
        CUST_CREDIT_LIMIT=10000                  RANGE SCAN          12
```

- The optimizer uses the B*-Tree index; however, this is less efficient compared to the corresponding bitmap index from the previous case in step 36.

39. **Case 10: Function Based Index:** Drop all the CUSTOMERS indexes, except its primary key index. Open a terminal window. Connect to sh and drop all the indexes that are currently

created on the CUSTOMERS table, except its primary key index, by executing drop_customers_indexes.sql.

```
$ cd /home/oracle/labs/solutions/Access_Paths/
$ sqlplus sh/sh

...
Connected to:...

SQL> @drop_customers_indexes.sql
```

40. After this, create one B*-tree index on the following column of the CUSTOMERS table:
cust_last_name (create_last_name_index.sql)

```
create_last_name_index.sql
SQL Worksheet History
Worksheet Query Builder
set echo on

CREATE INDEX cust_cust_last_name_idx
ON customers(cust_last_name)
NOLOGGING COMPUTE STATISTICS;

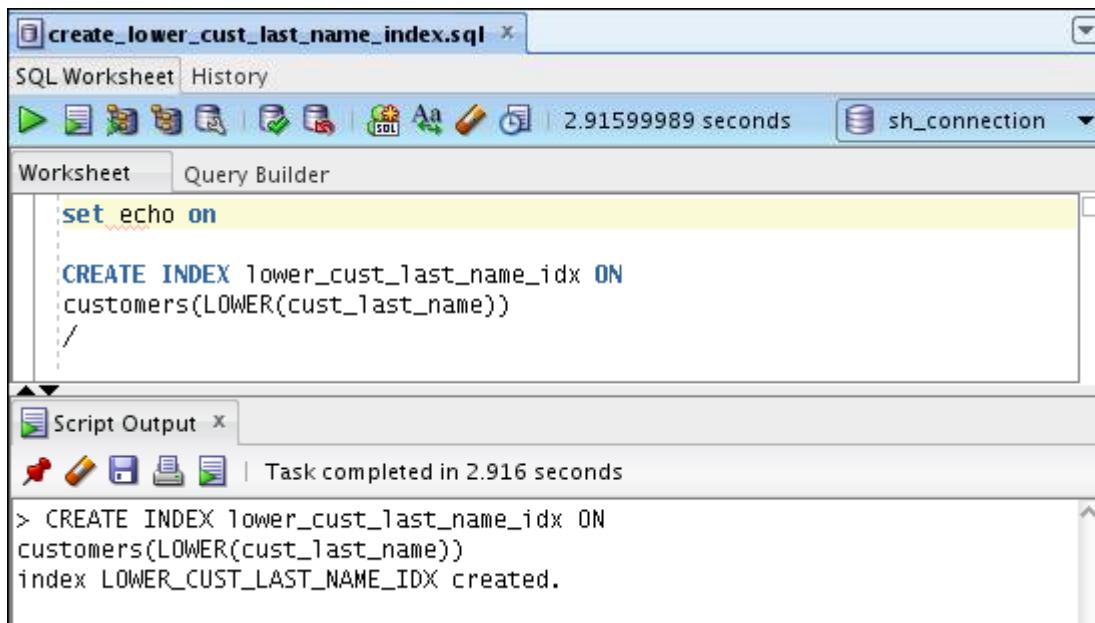
Script Output
Task completed in 1.041 seconds
> CREATE INDEX cust_cust_last_name_idx
ON customers(cust_last_name)
NOLOGGING COMPUTE STATISTICS
index CUST_CUST_LAST_NAME_IDX created.
```

41. Autotrace the query in query12.sql. What do you observe?

```
query12.sql
SQL Worksheet History
Worksheet Query Builder
SELECT cust_id, country_id
FROM customers
WHERE LOWER( cust_last_name) LIKE 'gentle'
/

Autotrace
SQL | 0.131 seconds
OPERATION          OBJECT_NAME      OPTIONS   COST
SELECT STATEMENT
  TABLE ACCESS      CUSTOMERS        FULL      423
    Filter Predicates
      LOWER(CUST_LAST_NAME)='gentle'
```

- Although there is an index, it cannot be used because its column is modified by a function.
42. How can you enhance the performance of the previous query without modifying the statement itself? Implement your solution.
- You can create a function-based index by using `create_lower_cust_last_name_index.sql`.



The screenshot shows the Oracle SQL Developer interface. The top window is titled "create_lower_cust_last_name_index.sql". The "Worksheet" tab is selected, displaying the SQL code:

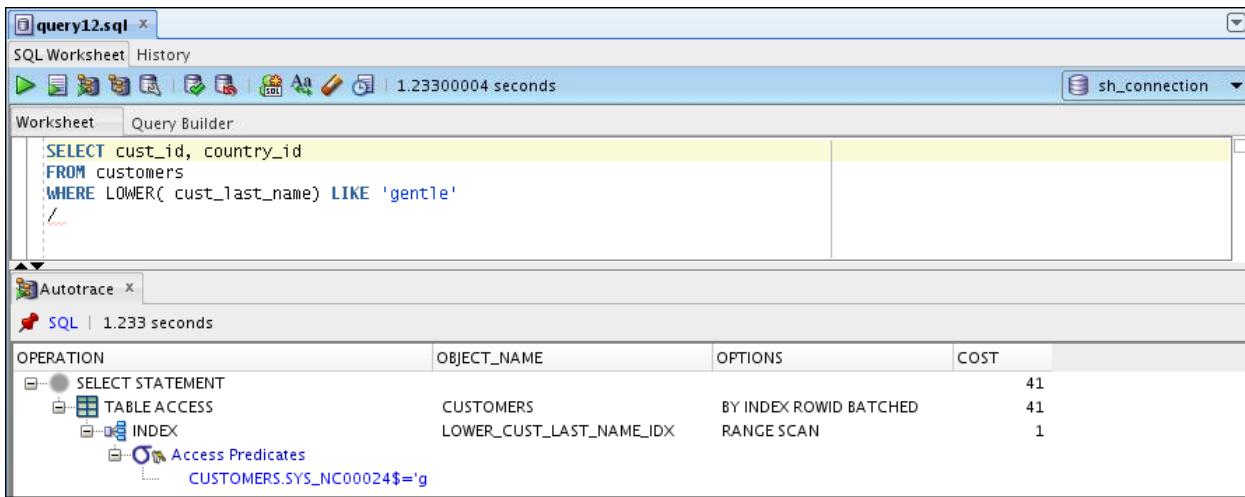
```
set echo on

CREATE INDEX lower_cust_last_name_idx ON
customers(LOWER(cust_last_name))
/
```

The "Script Output" tab at the bottom shows the execution results:

```
> CREATE INDEX lower_cust_last_name_idx ON
customers(LOWER(cust_last_name))
index LOWER_CUST_LAST_NAME_IDX created.
```

43. Check if your solution executes faster than in the case of the query in step 41.



The screenshot shows the Oracle SQL Developer interface. The top window is titled "query12.sql". The "Worksheet" tab is selected, displaying the SQL code:

```
SELECT cust_id, country_id
FROM customers
WHERE LOWER( cust_last_name) LIKE 'gentle'
/
```

The "Autotrace" tab at the bottom shows the execution plan:

| OPERATION | OBJECT_NAME | OPTIONS | COST |
|-------------------|-----------------------------|------------------------|------|
| SELECT STATEMENT | | | 41 |
| TABLE ACCESS | CUSTOMERS | BY INDEX ROWID BATCHED | 41 |
| INDEX | LOWER_CUST_LAST_NAME_IDX | RANGE SCAN | 1 |
| Access Predicates | CUSTOMERS.SYS_NC00024\$='g' | | |

44. **Case 11: Index Organized Table:** Execute the `iot_setup.sql` script by using the `sh` user to set up the environment for this case.

The screenshot shows the Oracle SQL Developer interface. In the top worksheet, a script named `iot_setup.sql` is running. It contains the following SQL code:

```

set echo on

CREATE table promotions_iot
(promo_id number primary key
, promo_name VARCHAR2(40)
, promo_subcategory VARCHAR2 (30)
, promo_category VARCHAR2 (30)
, promo_cost NUMBER
, promo_begin_date DATE
, promo_end_date DATE)
ORGANIZATION INDEX
/

INSERT INTO promotions_iot
SELECT promo_id, promo_name, promo_subcategory, promo_category, promo_cost, promo_begin_date, promo_end_date
FROM promotions
/

```

In the bottom worksheet, the output of the script execution is shown:

```

> CREATE table promotions_iot
(promo_id number primary key
, promo_name VARCHAR2(40)
, promo_subcategory VARCHAR2 (30)
, promo_category VARCHAR2 (30)
, promo_cost NUMBER
, promo_begin_date DATE
, promo_end_date DATE)
ORGANIZATION INDEX
table PROMOTIONS_IOT created.
> INSERT INTO promotions_iot
SELECT promo_id, promo_name, promo_subcategory, promo_category, promo_cost, promo_begin_date, promo_end_date
FROM promotions
503 rows inserted.

```

45. Autotrace the query in `query13a.sql` and `query13b.sql`. What do you observe?

The screenshot shows the Oracle SQL Developer interface. In the top worksheet, two files are open: `query13a.sql` and `query13b.sql`. The `query13a.sql` file contains the following SQL code:

```

SELECT *
FROM promotions
WHERE promo_id > 300
/

```

In the bottom worksheet, the output of the query execution is shown:

Autotrace

SQL | 0.157 seconds

| OPERATION | OBJECT_NAME | OPTIONS | COST |
|-------------------|-------------|---------|------|
| SELECT STATEMENT | PROMOTIONS | FULL | 17 |
| TABLE ACCESS | | | 17 |
| Filter Predicates | | | |
| PROMO_ID>300 | | | |

query13a.sql x query13b.sql x

SQL Worksheet History

Worksheet Query Builder

```
SELECT /*+ INDEX(promotions) */ *
FROM promotions
WHERE promo_id > 300
/
```

Autotrace x

SQL | 0.783 seconds

| OPERATION | OBJECT_NAME | OPTIONS | COST |
|-------------------|--------------|------------------------|------|
| SELECT STATEMENT | | | 353 |
| TABLE ACCESS | PROMOTIONS | BY INDEX ROWID BATCHED | 353 |
| INDEX | PROMO_PK | RANGE SCAN | 1 |
| Access Predicates | PROMO_ID>300 | | |

- The first query lets the optimizer decide the plan and the best that it can find is to perform a full table scan. Forcing the use of an index is not a good idea because it takes longer to execute.

46. Autotrace the query in query14.sql.

```
SELECT *
FROM promotions_iot
WHERE promo_id > 300;
```

What do you observe?

query13a.sql x query13b.sql x query14.sql x

SQL Worksheet History

Worksheet Query Builder

```
SELECT *
FROM promotions_iot
WHERE promo_id > 300
/
```

Autotrace x

SQL | 1.506 seconds

| OPERATION | OBJECT_NAME | OPTIONS | COST |
|-------------------|--------------------|------------|------|
| SELECT STATEMENT | | | 2 |
| INDEX | SYS_IOT_TOP_188472 | RANGE SCAN | 2 |
| Access Predicates | PROMO_ID>300 | | |

- The optimizer directly uses the index-organized structure, which is extremely efficient in this case compared to the previous step 46.

47. Open and execute the iot_cleanup.sql script to clean up your environment.

The screenshot shows the Oracle SQL Developer interface. The top tab bar has 'iot_cleanup.sql' selected. The main workspace shows the following SQL code:

```
set echo on
drop table promotions_iot purge;
```

The 'Script Output' pane at the bottom shows the results of the execution:

```
> drop table promotions_iot purge
table PROMOTIONS_IOT dropped.
```

48. **Case 12: Index Skip Scan:** Execute the iss_setup.sql script by using the sh user to set up your environment for this practice.

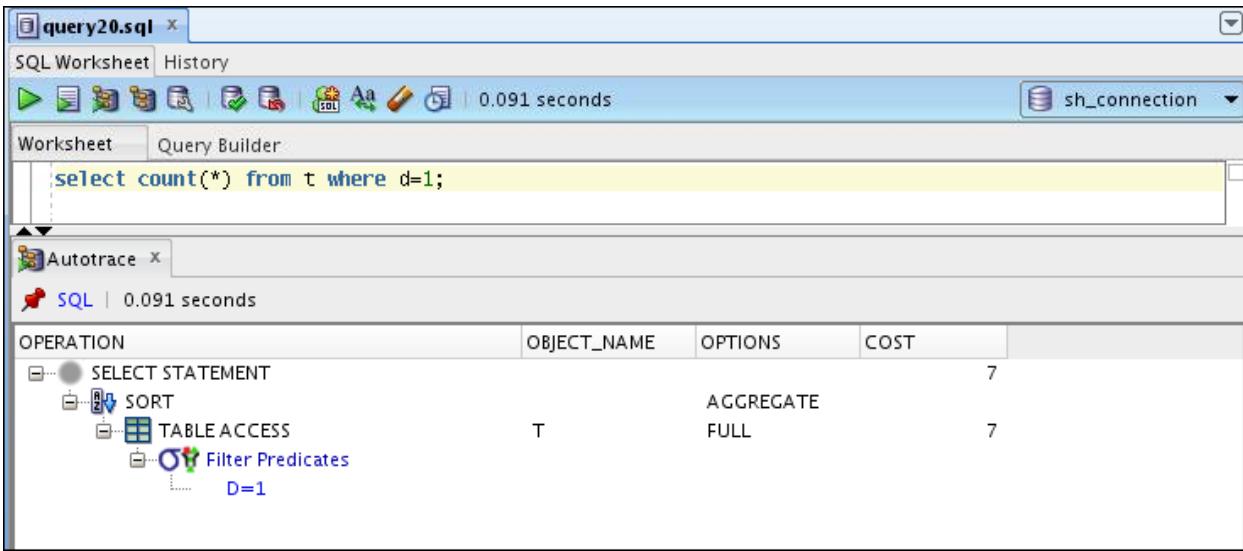
The screenshot shows the Oracle SQL Developer interface. The top tab bar has 'iss_setup.sql' selected. The main workspace shows the following PL/SQL code:

```
set echo on
create table t(c number, d number);
begin
  for i in 1..10000 loop
    insert into t values(1,i);
  end loop;
end;
/
create index it on t(c,d);
```

The 'Script Output' pane at the bottom shows the results of the execution:

```
> create table t(c number, d number)
table T created.
> begin
  for i in 1..10000 loop
    insert into t values(1,i);
  end loop;
end;
anonymous block completed
> create index it on t(c,d)
index IT created.
```

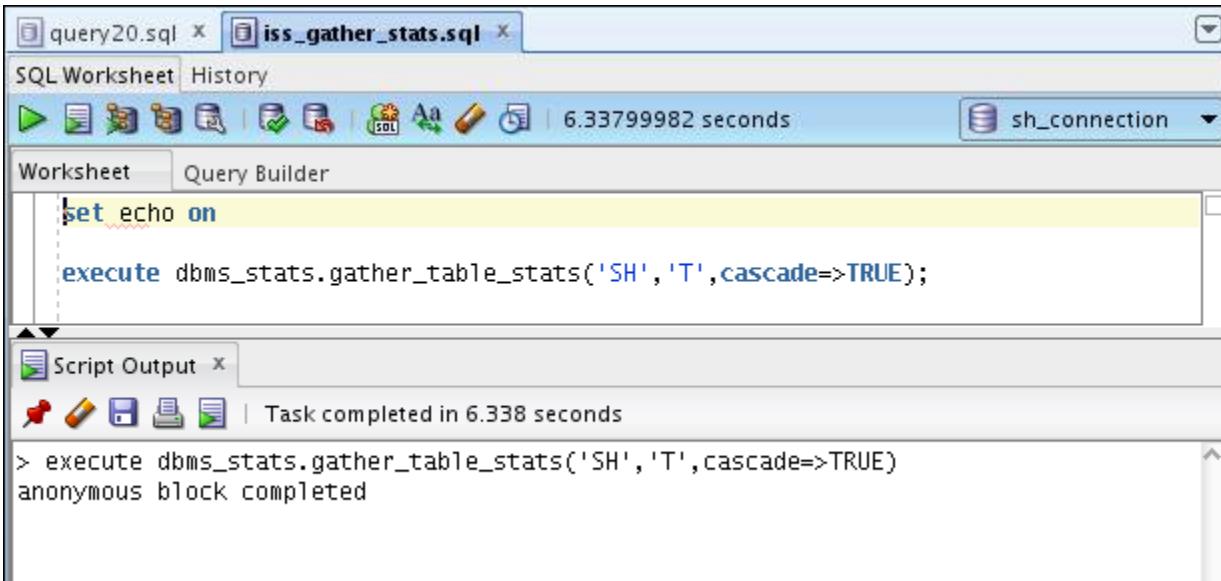
49. Autotrace the query in `query20.sql`. What do you observe?



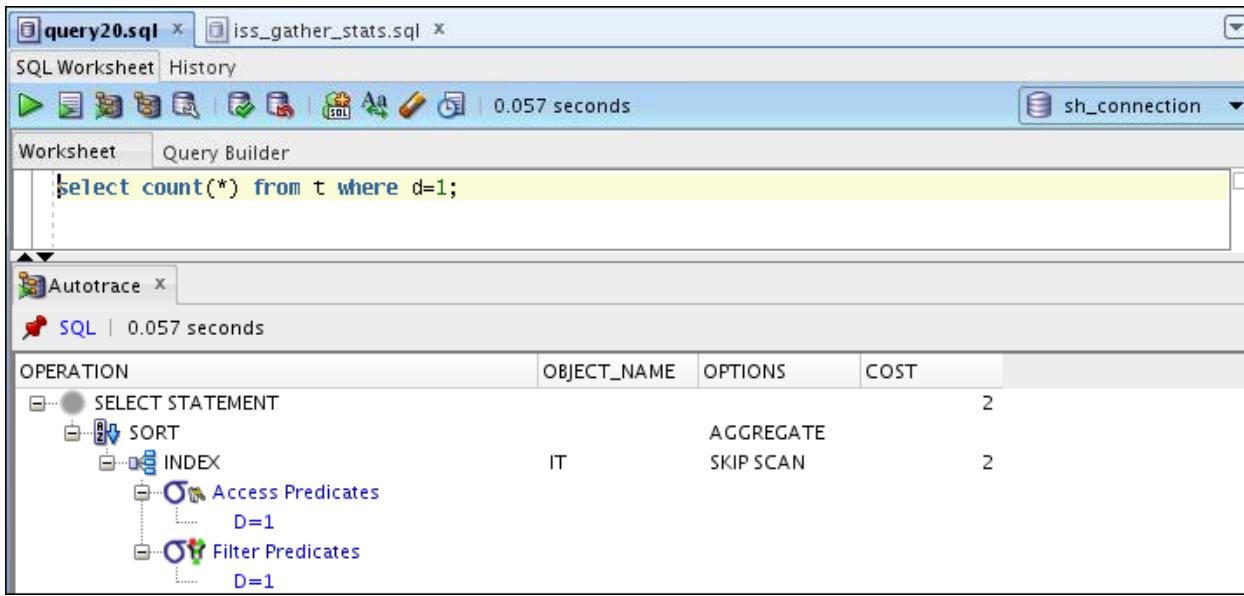
- The optimizer is not using the index and does a full table scan.

50. How would you improve the performance of a query, such as the one in the previous step? Implement your solution.

- a. Execute `iss_gather_stats.sql` to make sure that you gather the statistics for your table correctly so that the index skip scan can be used.



51. Autotrace the query in `query20.sql`. What do you observe?

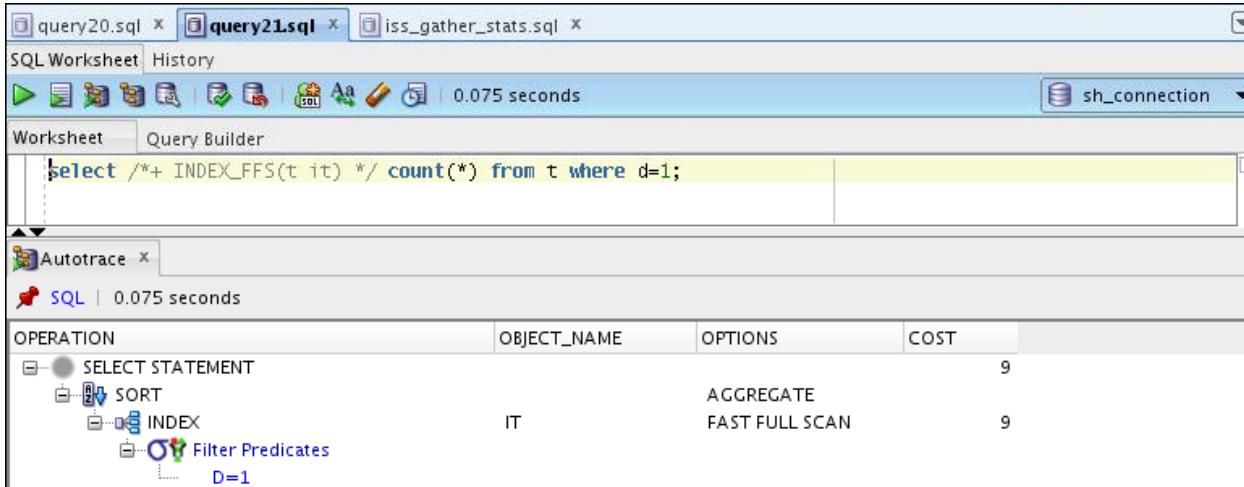


- The optimizer now uses the index to perform an index skip scan.

52. Compare the result of `query20.sql` with the result that you obtain when you autotrace the following query (`query21.sql`):

```
select /*+ INDEX_FFS(t it) */ count(*) from t where d=1;
```

What do you observe?



- The optimizer uses a fast full index scan, but this is not better than the index skip scan.

53. Execute the `iss_cleanup.sql` script to clean up your environment for this case.

The screenshot shows the Oracle SQL Worksheet interface. The title bar has three tabs: 'query20.sql' (closed), 'query21.sql' (closed), and 'iss_cleanup.sql'. The 'iss_cleanup.sql' tab is active. The toolbar includes icons for running, saving, and connecting. The status bar shows '1.30599999 seconds' and a connection named 'sh_connection'. The main area is a 'Worksheet' tab, containing the following SQL code:

```
set echo on
drop table t purge;
```

The output window below shows the results of the execution:

```
> drop table t purge
table T dropped.
```

Practices for Lesson 9: Optimizer: Join Operators

Chapter 9

Practices for Lesson 9: Overview

Practices Overview

In these practices, you will examine three SQL statements and compare the different join methods for each statement.

Practice 9: Using Join Paths

Overview

In this practice, you explore the various access paths that the optimizer can use and compare them. You have the possibility of exploring different scenarios, each of which is self-contained. All scripts needed for this practice can be found in your \$HOME/labs/solutions/Access_Paths directory.

Tasks

1. Open and execute the join_setup.sql script by using sys_connection.

The screenshot shows the Oracle SQL Worksheet interface. The title bar says 'sys_connection' and 'join_setup.sql'. The 'Worksheet' tab is selected. The code area contains the following SQL:

```
-- execute as sys user
grant select any dictionary to OE;
grant select_catalog_role to OE;
grant select any dictionary to SH;
grant select_catalog_role to SH;
exit;
```

The 'Script Output' pane at the bottom shows the results of the execution:

```
grant succeeded.
grant succeeded.
grant succeeded.
grant succeeded.
Commit
```

2. Open the connection named scott that you created in Practice 2. If you have not created this connection, create it now with the following attributes.

Create a connection with the following specifications:

Name: scott_connection

Username: scott

Password: tiger

Select Save Password.

SID: systun

Click Test.

Click Connect.

3. **Case 1: Nested Loops Join Path:** The nested loops join method works well with very small tables, and larger tables with indexes on the join column, and at least one table that produces a small row source. The nested loops method will produce joined rows as soon as

a match is found, so it is sometimes the preferred method for the FIRST_ROWS access goal.

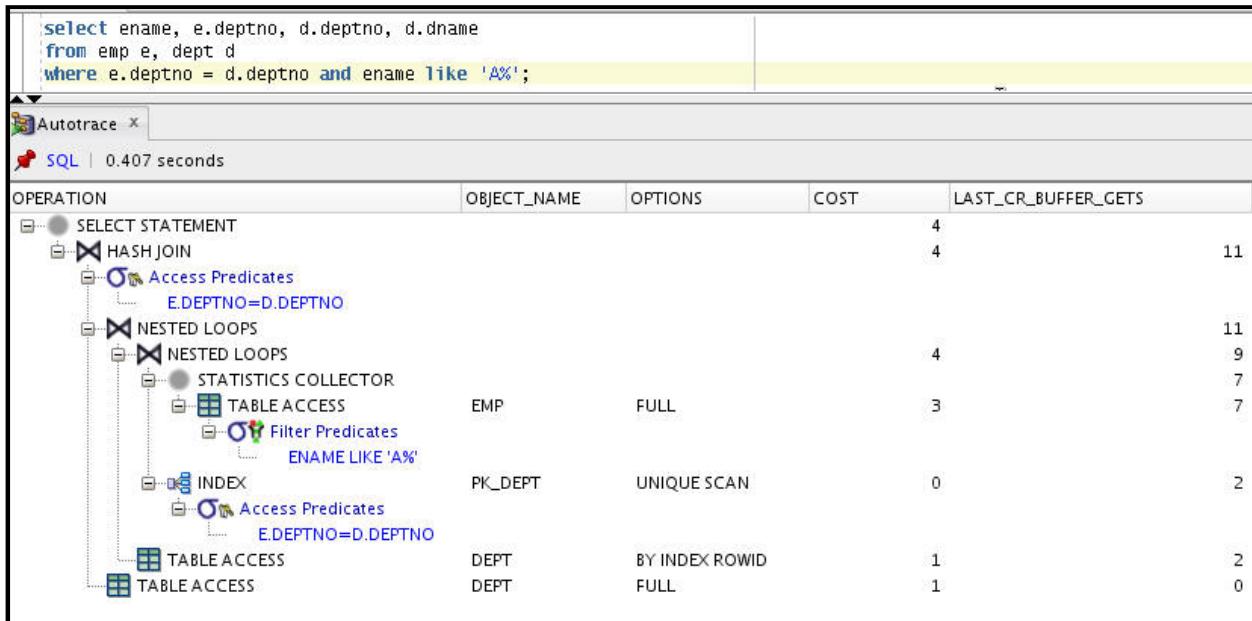
- Enter the following query or open `small_tables_join.sql` as the `scott` user:

```
select ename, e.deptno, d.deptno, d.dname
from emp e, dept d
where e.deptno = d.deptno and ename like 'A%';
```

Run Autotrace twice to load the buffer cache. Observe the statistics in the second run.

What do you observe?

- The optimizer chooses the nested loops join method for this query by default. The cost is low. This is expected because both tables in the query are very small.

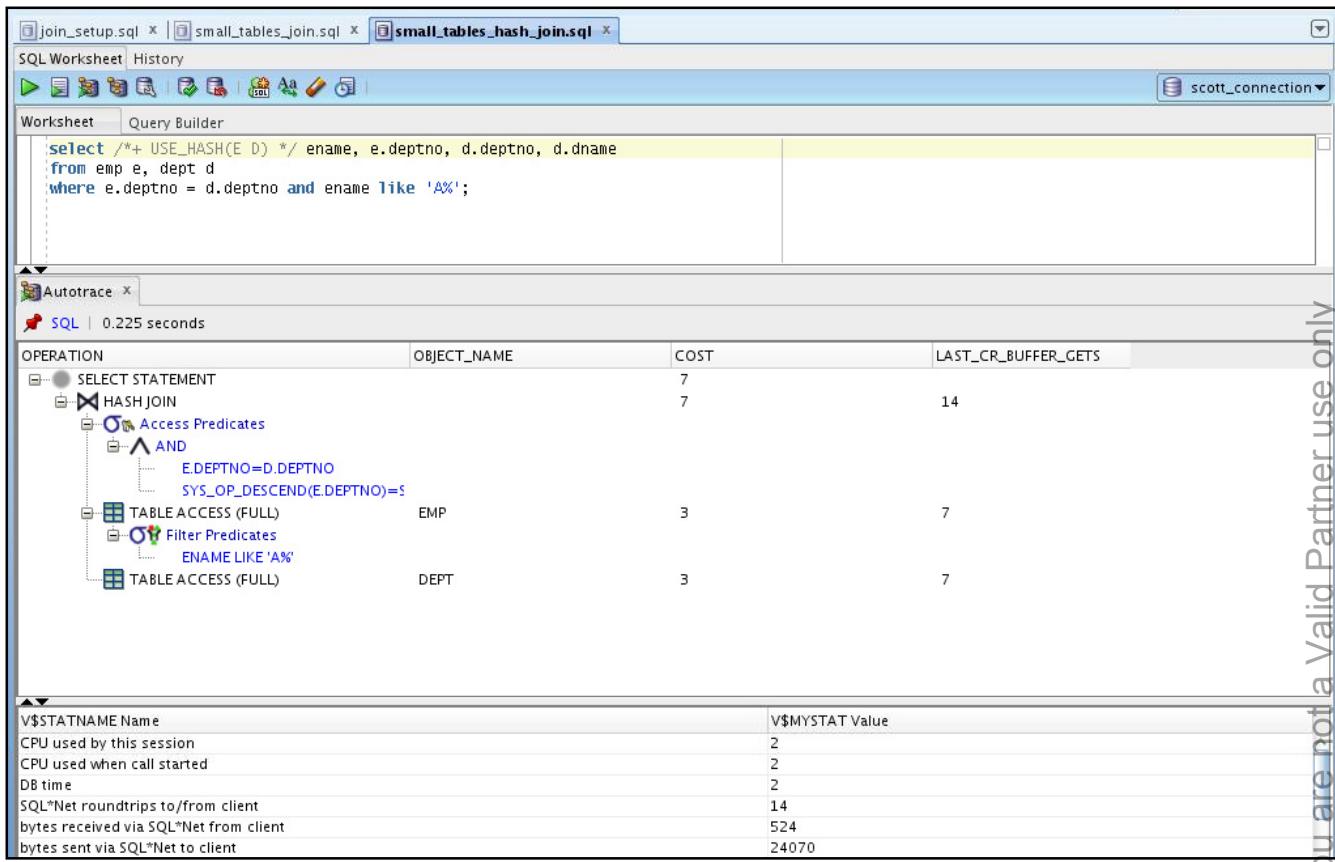


- Enter the same query in SQL Developer or open the `small_tables_hash_join.sql` script, but force the use of a hash join by using the following query with hints. Then use Autotrace to observe the differences in the execution plan and statistics.

```
select /*+ USE_HASH(E D) */ ename, e.deptno, d.deptno, d.dname
from emp e, dept d
where e.deptno = d.deptno and ename like 'A%';
```

What do you observe?

The optimizer is forced to use the hash join method. The number of consistent gets is higher and the estimated cost is higher. Note that the elapsed time may vary depending on any other activity on the machine.

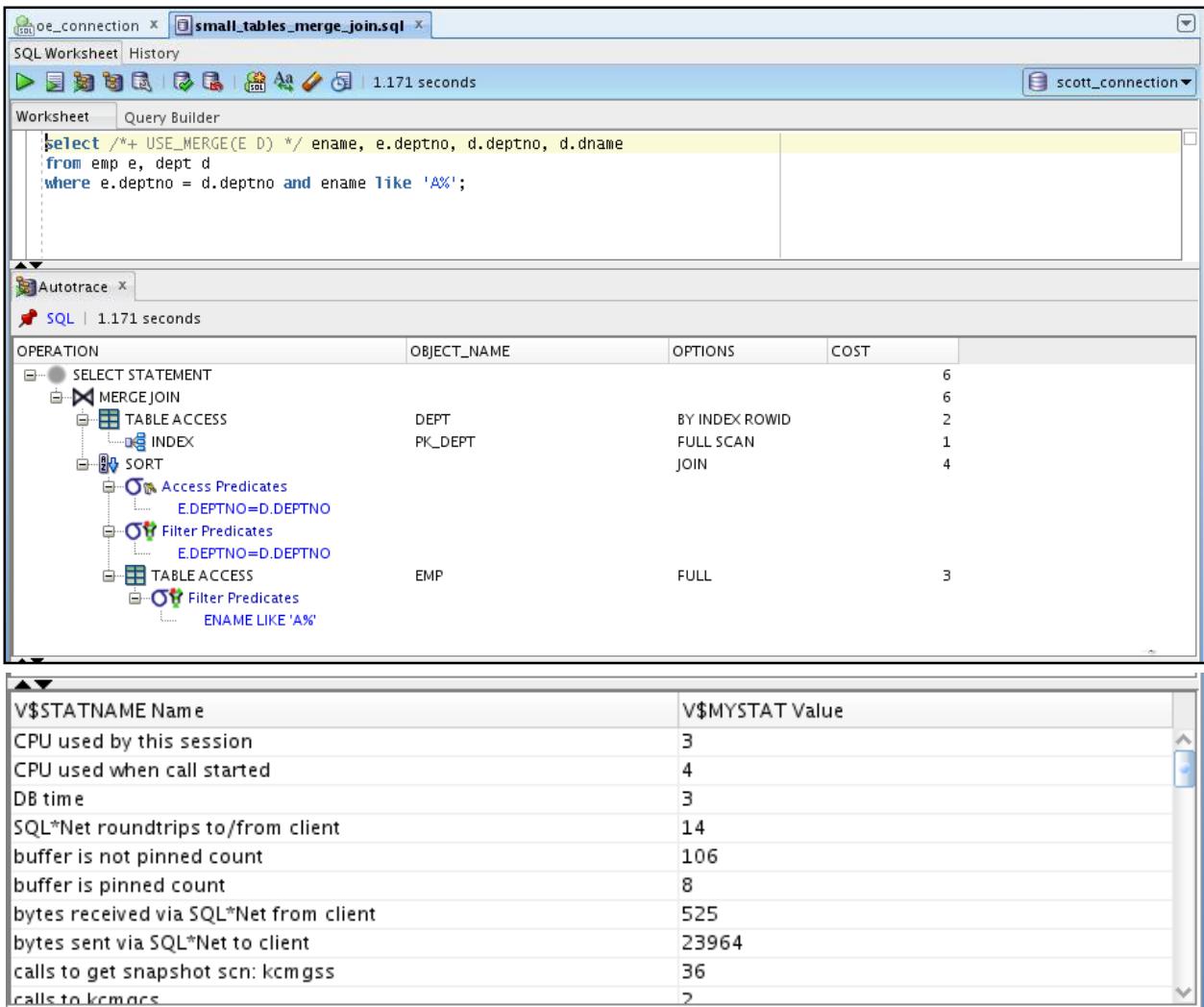


- d. Enter the same query in SQL Developer or open `small_tables_merge_join.sql`, but force the use of a sort-merge join by using the following query with hints. Then use Autotrace to observe the differences in the execution plan and statistics.

```
select /*+ USE_MERGE(E D) */ ename, e.deptno, d.deptno, d.dname
from emp e, dept d
where e.deptno = d.deptno and ename like 'A%';
```

What do you observe?

The optimizer is forced to use the sort-merge join method. The number of consistent gets is lower than nested loops, but there is the additional cost of the sorts that is not included in the consistent gets. The estimated cost is higher. Note that the elapsed time may vary depending on any other activity on the machine.



4. Open the connection named `oe_connection` that you created in Practice 2. If you have not created this connection, create it now with the following attributes.
 - a. Click the Add Connection button.
 - b. Create a connection with following specifications:
 - Name: `oe_connection`
 - Username: `oe`
 - Password: `oe`
 - Select Save Password.
 - Sid: `systun`
 - Click Test.
 - Click Connect.
5. **Case 2: Merge Join:** The merge join is a general-purpose join method that can work when other methods cannot. The merge join must sort each of the two row sources before performing the join. Because both row sources must be sorted before the first joined row is returned, the merge join method is not well suited for use with the `FIRST_ROWS` goal.

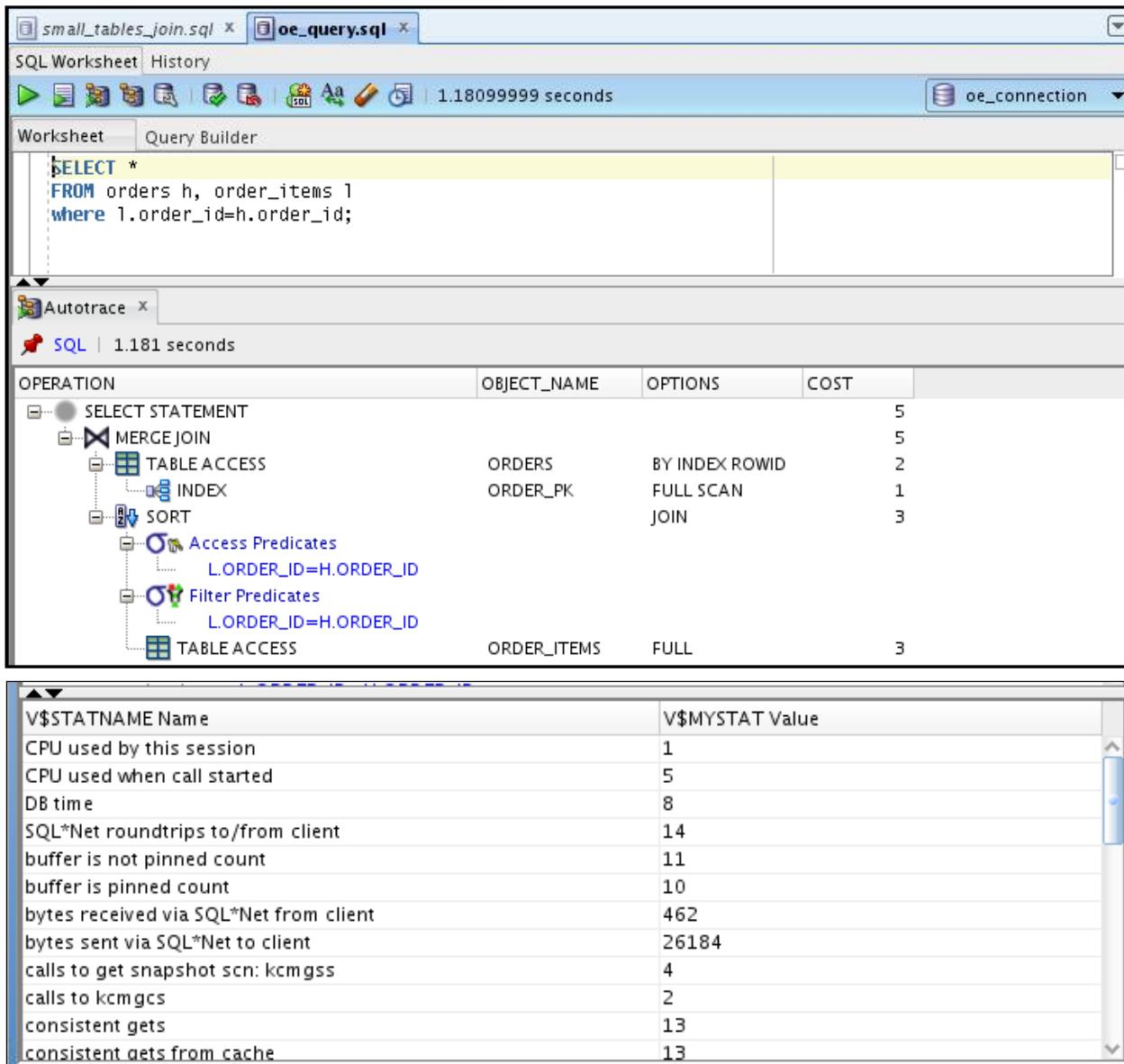
- a. Enter the following query in a SQL Developer worksheet connected as the OE user or open the oe_query.sql script.

```
SELECT *
  FROM orders h, order_items l
 WHERE l.order_id = h.order_id;
```

- b. Autotrace this query twice to warm the buffer cache.

What do you observe?

Notice that the merge join is chosen by the optimizer.



- c. Enter the same query with a hint to force the use of a hash join or open the oe_query_hash.sql script.

```
SELECT /*+ USE_HASH(h l) */ *
  FROM orders h, order_items l
 WHERE l.order_id = h.order_id;
```

What do you observe?

Compare the estimated cost to the merge join in the preceding step, and the consistent gets. In this case, the additional sort required by the merge join method does not raise the cost enough to cause the hash join to be chosen by the optimizer.

The screenshot shows the Oracle SQL Developer interface. At the top, there are two tabs: 'oe_query.sql' and 'oe_query_hash.sql'. The 'oe_query_hash.sql' tab is active, showing the following SQL code:

```
SELECT /*+ USE_HASH(h l) */ *
  FROM orders h, order_items l
 WHERE l.order_id = h.order_id;
```

Below the code, the status bar indicates '0.64600003 seconds'. The main workspace is titled 'Worksheet' and contains the Autotrace output. The output shows the execution plan:

| OPERATION | OBJECT_NAME | OPTIONS | COST |
|-------------------|-----------------------|---------|------|
| SELECT STATEMENT | | | 6 |
| HASH JOIN | | | 6 |
| Access Predicates | L.ORDER_ID=H.ORDER_ID | | |
| TABLE ACCESS | ORDERS | FULL | 3 |
| TABLE ACCESS | ORDER_ITEMS | FULL | 3 |

At the bottom of the interface, there is a table titled 'V\$STATNAME' with columns 'Name' and 'Value'.

| V\$STATNAME Name | V\$MYSTAT Value |
|--|-----------------|
| CPU used by this session | 2 |
| CPU used when call started | 2 |
| DB time | 2 |
| SQL*Net roundtrips to/from client | 14 |
| buffer is not pinned count | 8 |
| bytes received via SQL*Net from client | 485 |
| bytes sent via SQL*Net to client | 27093 |
| calls to get snapshot scn: kcmgss | 5 |
| calls to kcmgcs | 8 |
| consistent gets | 23 |
| consistent gets from cache | 23 |

- d. Enter the same query again, changing the hint to force the use of nested loops, or open the oe_query_nl.sql script.

```
SELECT /*+ USE_NL(h l) */ *
  FROM orders h, order_items l
 WHERE l.order_id = h.order_id;
```

Autotrace this query.

What do you observe?

Notice the estimated cost and the consistent gets. The cost of the nested loops join method is much higher than either the hash join method or the merge join method. The

nested loops method depends on either very small tables or indexes to be an efficient access path.

The screenshot shows two windows in Oracle SQL Developer:

- Autotrace Results:** A tree view of the execution plan for a query. The root node is "SELECT STATEMENT". It branches into "NESTED LOOPS", which further branches into "TABLE ACCESS" for the "ORDERS" table and "TABLE ACCESS" for the "ORDER_ITEMS" table. A "Filter Predicates" node is shown under the nested loops. The "OPTIONS" column shows "FULL" for both table access nodes, and the "COST" column shows values of 148, 148, 3, and 1 respectively.
- V\$STATNAME Statistics:** A table showing session statistics. The columns are "V\$STATNAME Name" and "V\$MYSTAT Value". The data includes:

| V\$STATNAME Name | V\$MYSTAT Value |
|--|-----------------|
| CPU used by this session | 1 |
| CPU used when call started | 1 |
| DB time | 2 |
| SQL*Net roundtrips to/from client | 14 |
| buffer is not pinned count | 8 |
| buffer is pinned count | 1 |
| bytes received via SQL*Net from client | 483 |
| bytes sent via SQL*Net to client | 26248 |
| calls to get snapshot scn: kcmgss | 5 |
| calls to kcmgcs | 20 |
| consistent gets | 65 |

6. Open the connection named `sh_connection` that you created in Practice 2. If you have not created this connection, create it now with the following attributes.
 - a. Click the Add Connection button.
 - b. Create a connection with the following specifications:
 - Name: `sh_connection`
 - Username: `sh`
 - Password: `sh`
 - Select Save Password.
 - Sid: `systun`
 - Click Test.
 - Click Connect.

7. **Case 3: Hash Join:** The hash join method performs very well on large row sources and on row sources where one row source is smaller. The hash join builds a hash table in memory (and overflows to the temp tablespace if the row source is too large) on the smaller of the row sources. The procedure then reads the second row source, probing for the hash value in the first. Because the rows are joined as soon as a match is found, the hash join method is also preferred for the FIRST_ROWS goal.

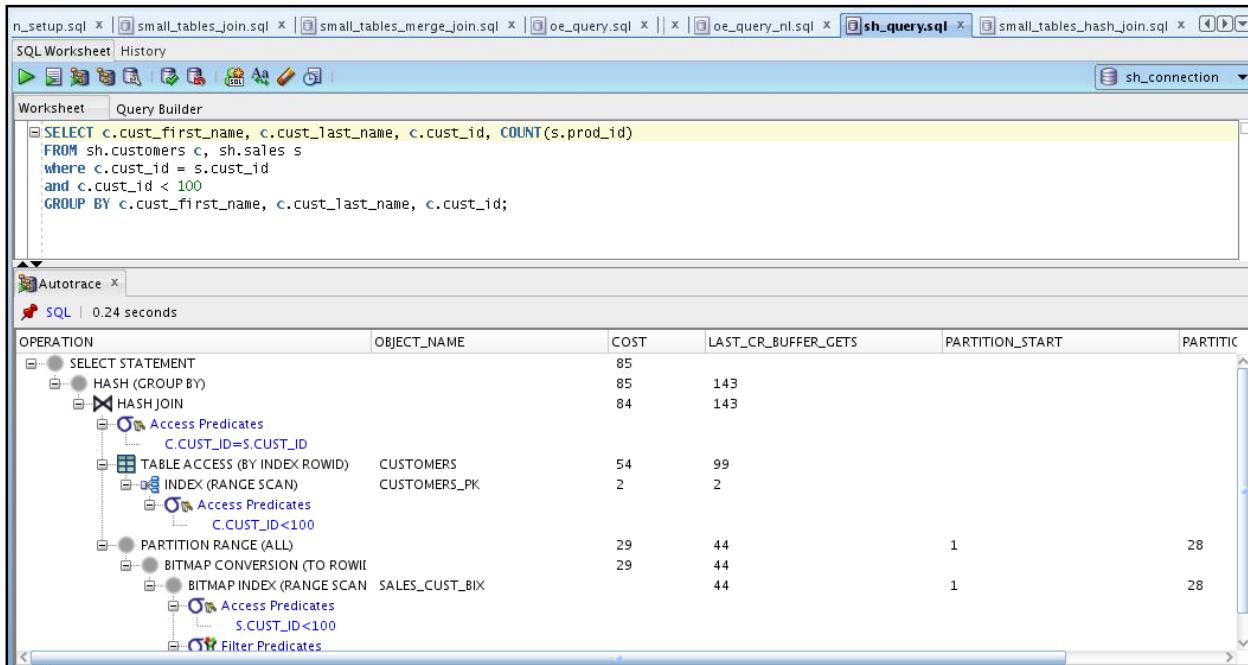
- a. Enter the following query in SQL Developer or open sh_query.sql:

```
SELECT c.cust_first_name, c.cust_last_name, c.cust_id,
COUNT(s.prod_id)
FROM sh.customers c, sh.sales s
where c.cust_id = s.cust_id
and c.cust_id < 100
GROUP BY c.cust_first_name, c.cust_last_name, c.cust_id;
```

Run Autotrace twice to warm the buffer cache.

What do you observe?

Notice the consistent gets and the estimated cost.



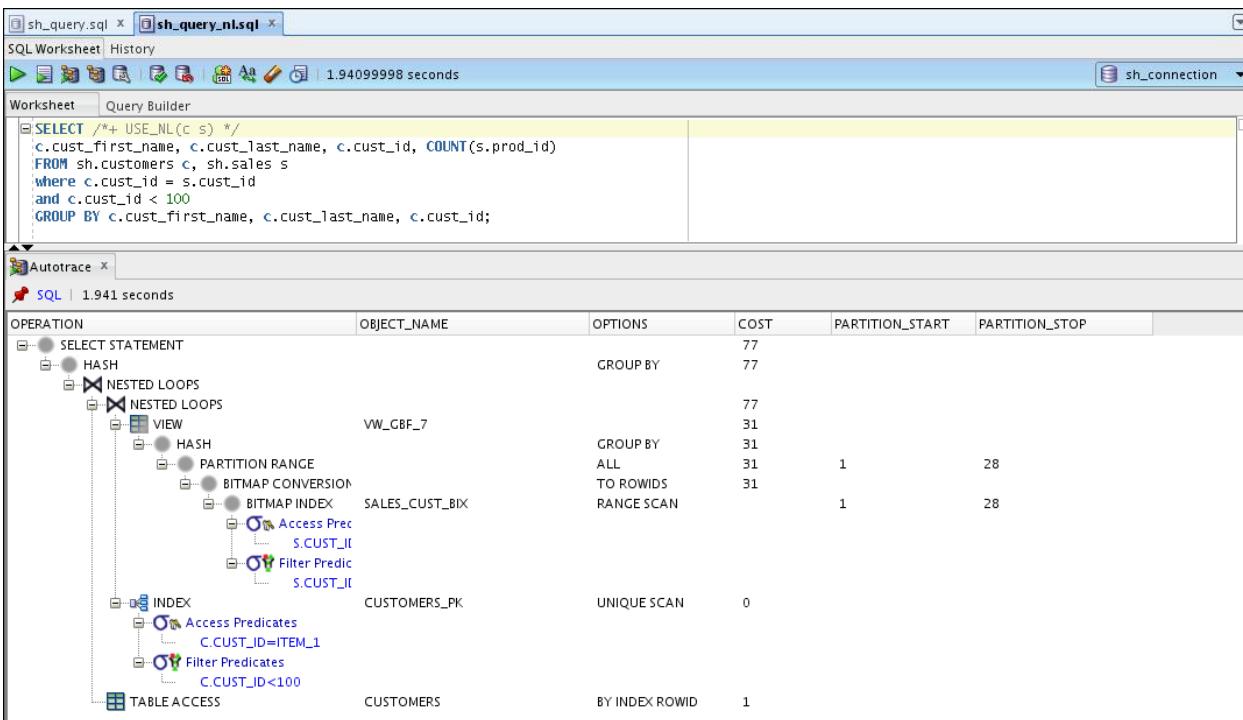
| V\$STATNAME Name | V\$MYSTAT Value |
|--|-----------------|
| CPU used by this session | 7 |
| CPU used when call started | 7 |
| Cached Commit SCN referenced | 94 |
| Commit SCN cached | 1 |
| DB time | 34 |
| HSC Heap Segment Block Changes | 1 |
| SQL*Net roundtrips to/from client | 14 |
| buffer is not pinned count | 282 |
| buffer is pinned count | 159 |
| bytes received via SQL*Net from client | 605 |
| bytes sent via SQL*Net to client | 25202 |

- b. Enter the same query again, but use a hint to force the nested loops method or open the sh_query_nl.sql script.

```
SELECT /*+ USE_NL(c s) */
       c.cust_first_name, c.cust_last_name,
       c.cust_id, COUNT(s.prod_id)
  FROM sh.customers c, sh.sales s
 WHERE c.cust_id = s.cust_id
   AND c.cust_id < 100
 GROUP BY c.cust_first_name, c.cust_last_name, c.cust_id;
```

Run Autotrace. What do you observe?

The consistent gets and estimated cost values are much higher than the values for the hash join method.



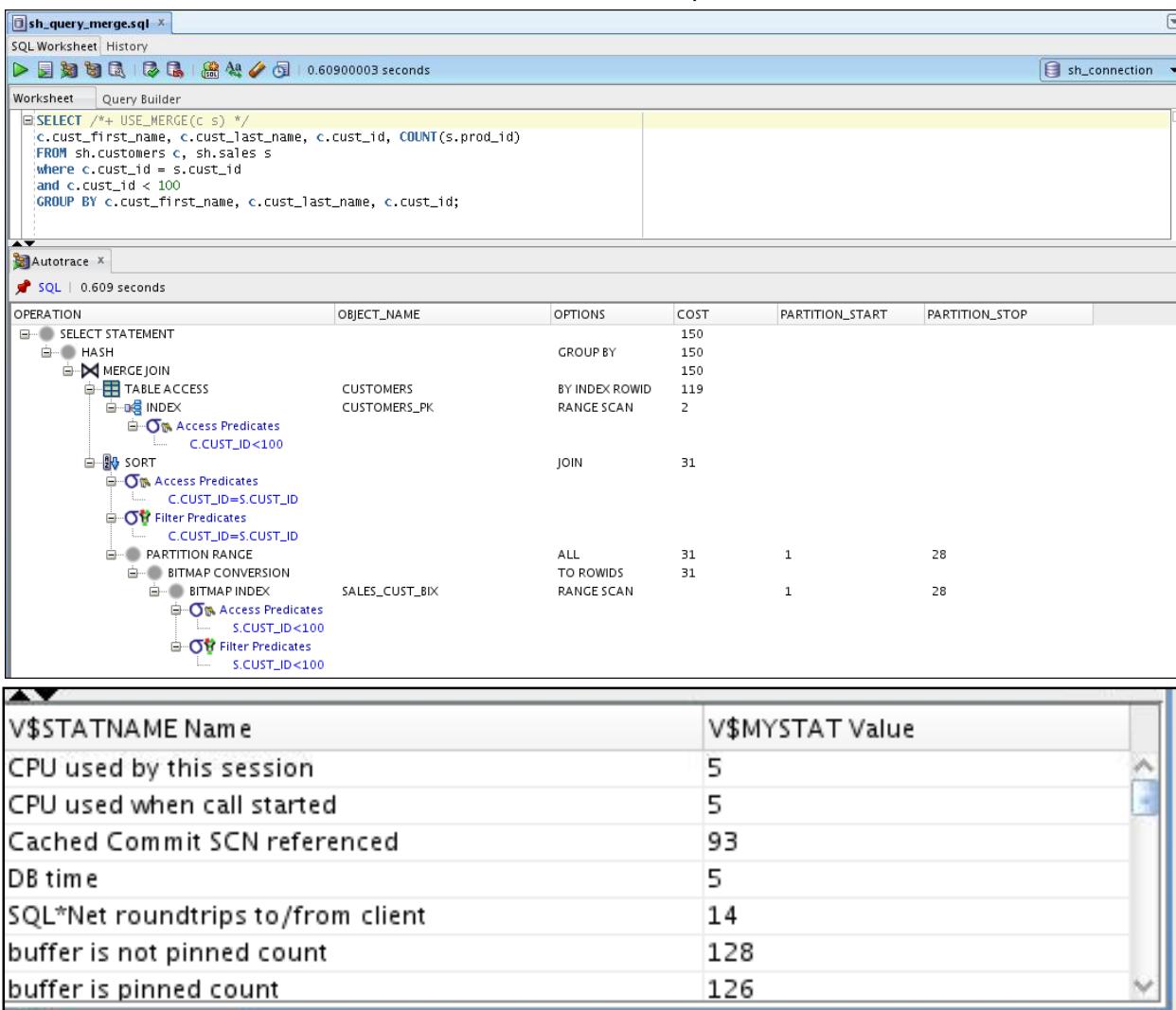
| V\$STATNAME Name | V\$MYSTAT Value |
|--|-----------------|
| CPU used by this session | 9 |
| CPU used when call started | 9 |
| Cached Commit SCN referenced | 5287 |
| DB time | 8 |
| SQL*Net roundtrips to/from client | 14 |
| buffer is not pinned count | 10959 |
| buffer is pinned count | 29 |
| bytes received via SQL*Net from client | 625 |
| bytes sent via SQL*Net to client | 25342 |
| calls to get snapshot scn: kcmgss | 3 |
| consistent gets | 5518 |
| .. | 5470 |

- c. Enter the same query again. This time use a hint to force the merge join method or open the sh_query_merge.sql script.

```
SELECT /*+ USE_MERGE(c s) */
c.cust_first_name, c.cust_last_name,
c.cust_id, COUNT(s.prod_id)
FROM sh.customers c, sh.sales s
where c.cust_id = s.cust_id
and c.cust_id < 100
GROUP BY c.cust_first_name, c.cust_last_name, c.cust_id;
```

Run Autotrace. What do you observe?

The consistent gets and the estimated cost values are higher than the hash join method, but still much lower than the values for the nested loops method.



THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Error : You are not a Valid Partner use only

Practices for Lesson 10: Other Optimizer Operators

Chapter 10

Practices for Lesson 10: Overview

Practices Overview

In these practices, you make use of the Results Cache to improve SQL performance with repeated queries and examine the SQL statements that use other access paths.

Practice 10-1: Using the Result Cache

In this practice, you explore the various possibilities of caching query results in the System Global Area (SGA). Perform the following steps to understand the use of Query Result Cache. All the scripts for this practice can be found in the \$HOME/labs/solutions/Query_Result_Cache/ directory.

1. The result_cache_setup.sh script was executed in the setup of this practice as the SYS user to create and populate the QRC schema. This script is listed as follows:

```
#!/bin/bash

cd /home/oracle/labs/solutions/Query_Result_Cache

sqlplus / as sysdba <<FIN!

set echo on

drop user qrc cascade;

create user qrc identified by qrc
default tablespace users
temporary tablespace temp;

grant connect, resource, dba to qrc;

connect qrc/qrc

exec dbms_result_cache.flush;

drop table cachejfv purge;

create table cachejfv(c varchar2(500)) tablespace users;

insert into cachejfv
values('aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa');

insert into cachejfv select * from cachejfv;
```

```
insert into cachejfv select * from cachejfv;
insert into cachejfv values('b');

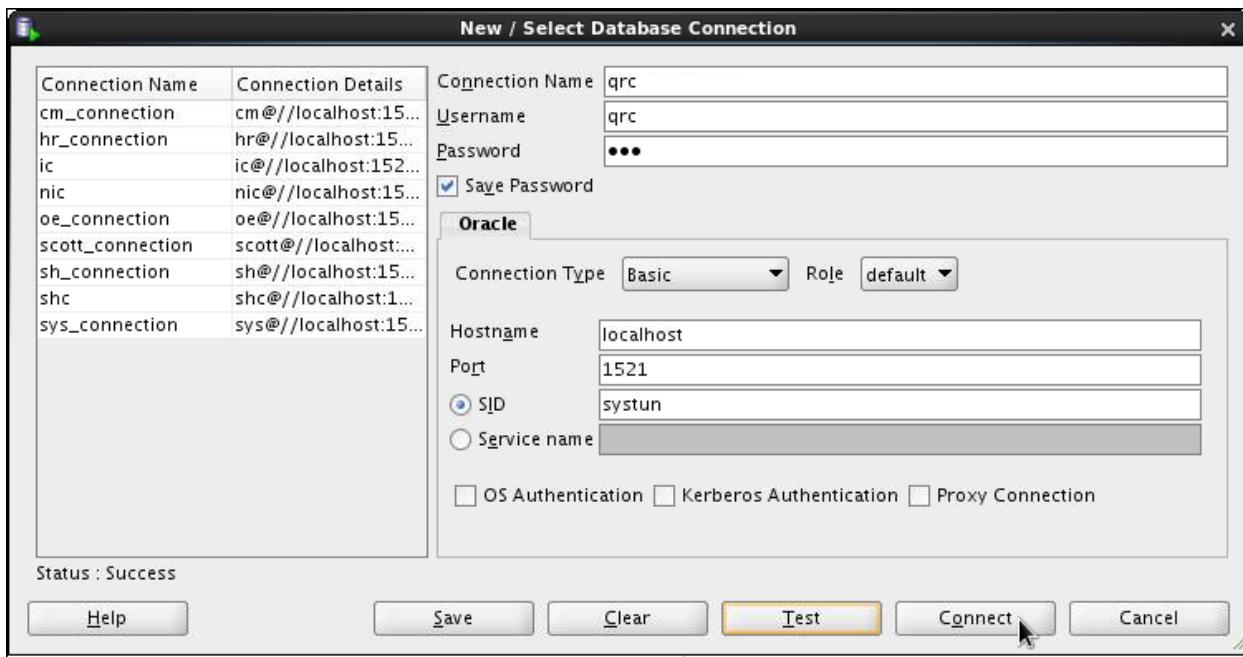
commit;

alter system flush buffer_cache;

FIN!

$
```

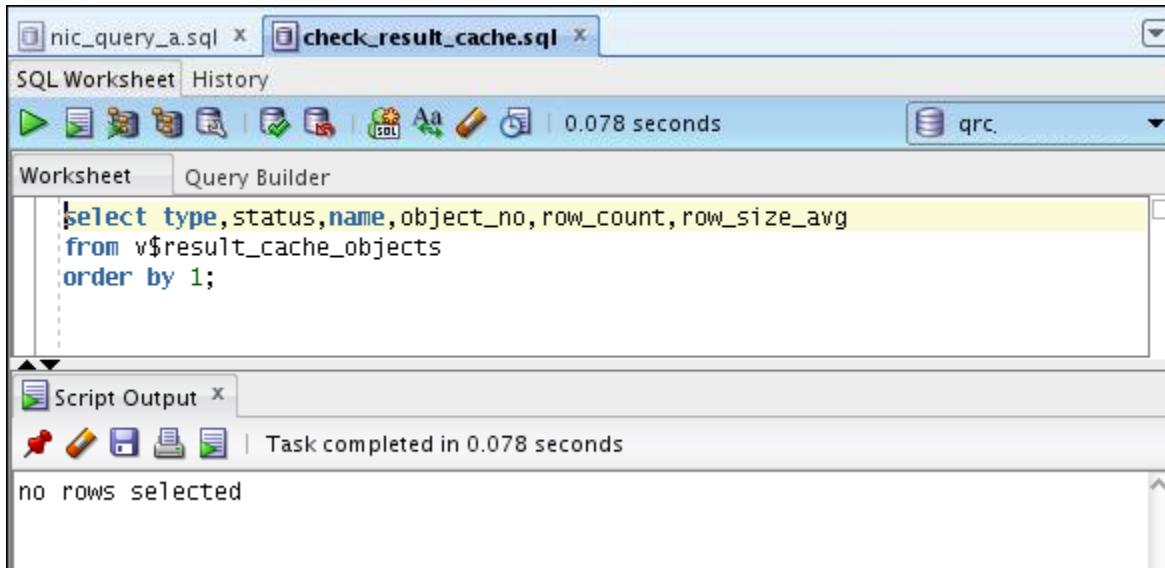
2. Create a connection for the `qrc` user.
 - a. Click the New Connection button.
 - b. Create a connection with the following specifications:
Name: `qrc`
Username: `qrc`
Password: `qrc`
Select “Save Password.”
Sid: `systun`
Click Test.
Click Connect.



3. As the `qrc` user, determine the current content of the query cache by using the following statement or open and execute the statement in the `check_result_cache.sql` file in the `$HOME/labs/solutions/Query_Result_Cache/` directory. From now on, execute all scripts as the `qrc` user.

```
select type,status,name,object_no,row_count,row_size_avg
from v$result_cache_objects order by 1;
```

What do you observe?



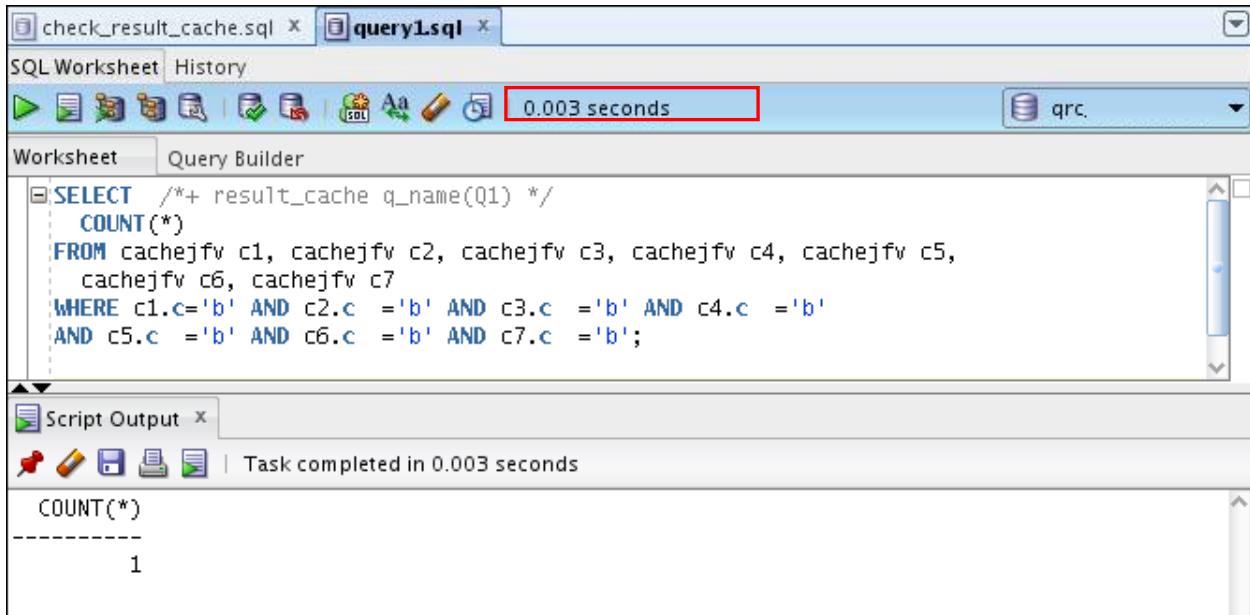
- No row is selected because the query cache is empty.

Note: If the result shows some rows then execute the following query using sys user and then execute the `check_result_cache.sql` script once again using `qrc` user.

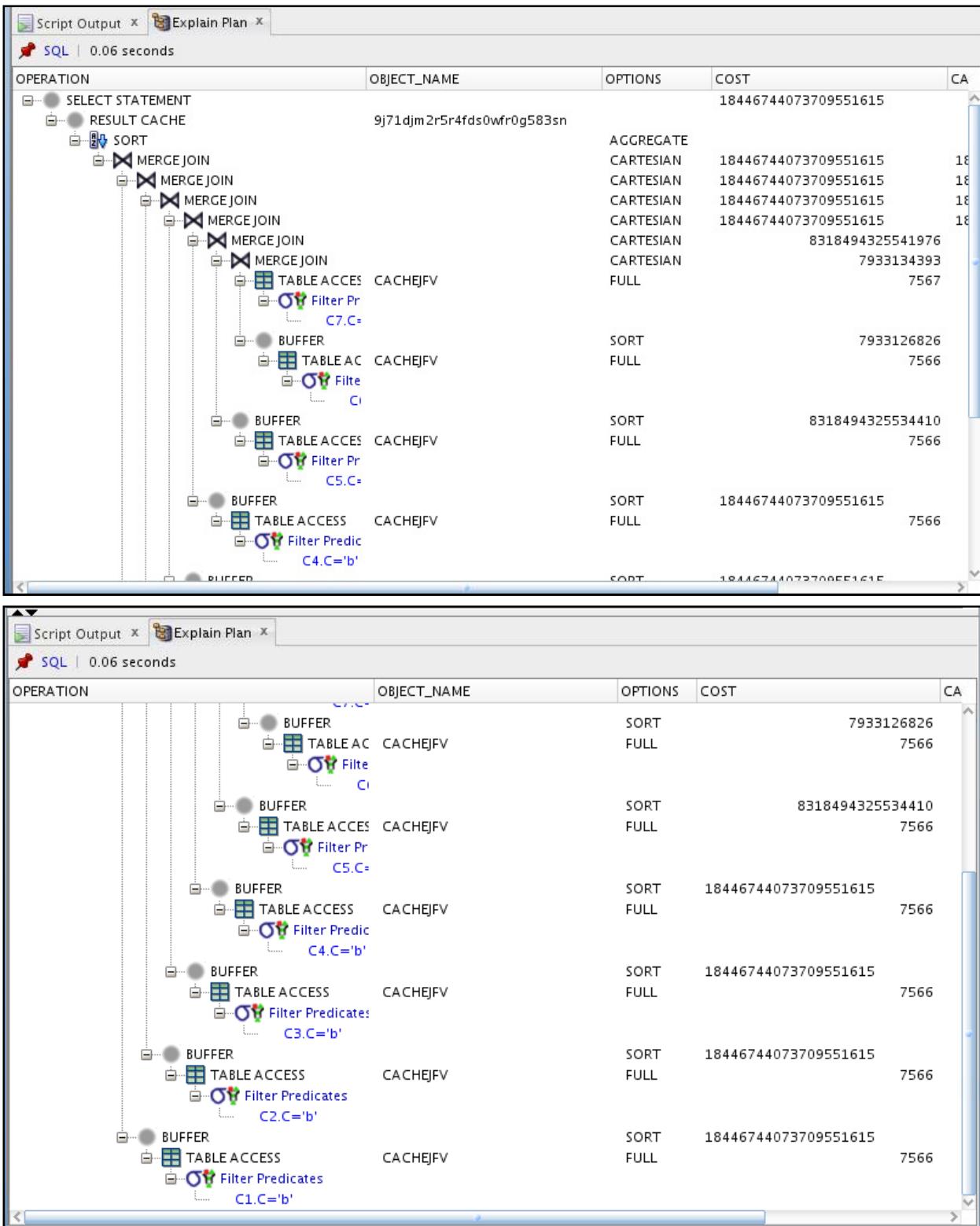
```
begin
  dbms_result_cache.flush;
end;
```

4. Open and execute the query1.sql script. Note the time it takes for this statement to execute.

```
SELECT /*+ result_cache q_name(Q1) */
  COUNT(*)
FROM cachejfv c1, cachejfv c2, cachejfv c3, cachejfv c4,
  cachejfv c5, cachejfv c6, cachejfv c7
WHERE c1.c='b' AND c2.c  ='b' AND c3.c  ='b' AND c4.c  ='b'
AND c5.c  ='b' AND c6.c  ='b' AND c7.c  ='b';
```



5. Determine the execution plan of the query in query1.sql. What do you observe?



- Because of the `result_cache` hint, the result of the query is computed using the result cache.

- As the `qrc` user, determine the current content of the query cache by using the following statement or the `check_result_cache.sql` script. What do you observe?

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

```
select type,status,name,object_no,row_count,row_size_avg
from v$result_cache_objects order by 1;
```

The screenshot shows the Oracle SQL Developer interface. In the top-left corner, there are tabs for 'check_result_cache.sql' and 'query1.sql'. The 'Worksheet' tab is active, displaying the SQL query: 'select type,status,name,object_no,row_count,row_size_avg from v\$result_cache_objects order by 1;'. Below the query, the 'Script Output' tab shows the results of the query execution. The output includes a header row with columns 'TYPE', 'STATUS', and 'NAME'. Under the 'Result' section, it shows a single row: 'Published SELECT /*+ result_cache q_name(Q1) */ COUNT(*) FROM cachejfv c1, cachejfv c2, cachejfv c3, cachejfv c4, cachejfv c5, cachej'. The status is listed as 'Published'.

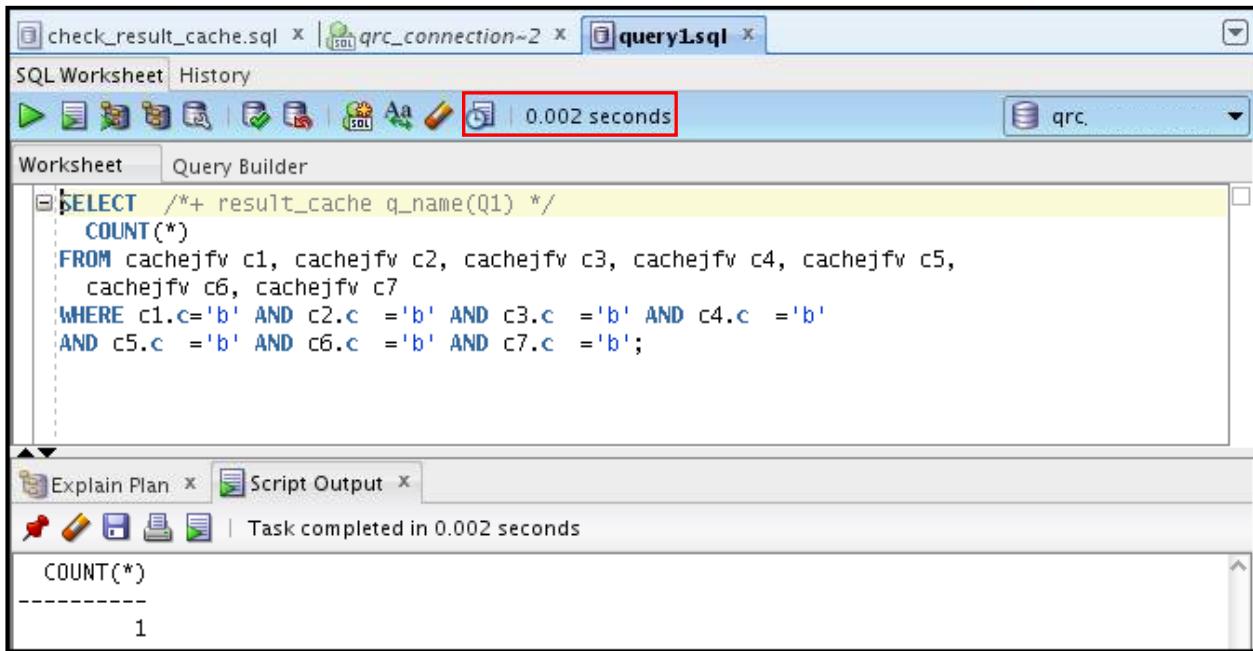
- You can now see that the result of your query is cached.

7. Flush the buffer cache of your instance.

```
alter system flush buffer_cache;
```

The screenshot shows the Oracle SQL Developer interface again. The 'Worksheet' tab has the command 'alter system flush buffer_cache;' highlighted. The 'Script Output' tab shows the execution results: 'system FLUSH altered.'.

8. Rerun query1.sql. What do you observe?



A screenshot of the Oracle SQL Developer interface. The title bar shows three tabs: 'check_result_cache.sql', 'qrc_connection~2', and 'query1.sql'. The 'query1.sql' tab is active. The toolbar has several icons, and the status bar at the bottom says '0.002 seconds'. The main area is a 'Worksheet' tab showing the following SQL code:

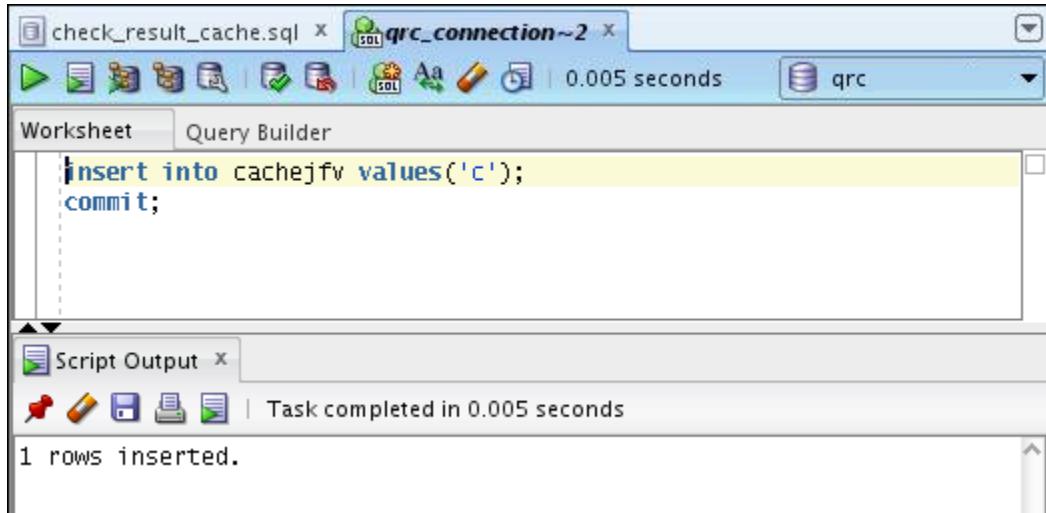
```
SELECT /*+ result_cache q_name(Q1) */  
  COUNT(*)  
FROM cachejfv c1, cachejfv c2, cachejfv c3, cachejfv c4, cachejfv c5,  
  cachejfv c6, cachejfv c7  
WHERE c1.c='b' AND c2.c = 'b' AND c3.c = 'b' AND c4.c = 'b'  
AND c5.c = 'b' AND c6.c = 'b' AND c7.c = 'b';
```

Below the worksheet, there are two panes: 'Explain Plan' and 'Script Output'. The 'Script Output' pane shows the output: 'Task completed in 0.002 seconds' followed by the result of the COUNT(*) query: '1'.

- The execution time for the query is now almost instantaneous.

9. Insert a new row into the CACHEJFV table by using the following statement:

```
insert into cachejfv values('c');  
commit;
```



A screenshot of the Oracle SQL Developer interface. The title bar shows three tabs: 'check_result_cache.sql', 'qrc_connection~2', and 'query1.sql'. The 'query1.sql' tab is active. The toolbar has several icons, and the status bar at the bottom says '0.005 seconds'. The main area is a 'Worksheet' tab showing the following SQL code:

```
insert into cachejfv values('c');  
commit;
```

Below the worksheet, there are two panes: 'Script Output' and 'Script Output'. The 'Script Output' pane shows the output: 'Task completed in 0.005 seconds' followed by the message '1 rows inserted.'

10. As the qrc user, determine the current content of the query cache by using the following statement or the check_result_cache.sql script. What do you observe?

```
select type,status,name,object_no,row_count,row_size_avg  
from v$result_cache_objects order by 1;
```

```

SELECT type, status, name, object_no, row_count, row_size_avg
FROM v$result_cache_objects
ORDER BY 1;

```

| TYPE | STATUS | NAME |
|------------|-----------|---|
| Dependency | Published | QRC.CACHEJFV |
| Result | inval | SELECT /*+ result_cache q_name(Q1) */ COUNT(*) FROM cachejfv c1, cachejfv c2, cachejfv c3, cachejfv c4, cachejfv c5, cachejfv c6, cachejfv c7 WHERE c1.c='b' AND c2.c = 'b' AND c3.c = 'b' AND c4.c = 'b' AND c5.c = 'b' AND c6.c = 'b' AND c7.c = 'b'; |

- The corresponding result cache entry is automatically invalidated.

11. Rerun query1.sql.

```

SELECT /*+ result_cache q_name(Q1) */
COUNT(*)
FROM cachejfv c1, cachejfv c2, cachejfv c3, cachejfv c4, cachejfv c5,
cachejfv c6, cachejfv c7
WHERE c1.c='b' AND c2.c = 'b' AND c3.c = 'b' AND c4.c = 'b'
AND c5.c = 'b' AND c6.c = 'b' AND c7.c = 'b';

```

| COUNT(*) |
|----------|
| 1 |

12. As the `qrc` user, determine the current content of the query cache by using the following statement or execute the `check_result_cache.sql` script. What do you observe?

```

select type, status, name, object_no, row_count, row_size_avg
from v$result_cache_objects order by 1;

```

The screenshot shows the Oracle SQL Worksheet interface. In the top tab bar, there are three tabs: 'check_result_cache.sql' (active), 'query1.sql', and 'qrc'. The main workspace contains a SQL query:

```
SELECT type, status, name, object_no, row_count, row_size_avg
FROM v$result_cache_objects
ORDER BY 1;
```

The results are displayed in a table:

| TYPE | STATUS | NAME |
|------------|-----------|--|
| Dependency | Published | QRC.CACHEJFV |
| Result | Invalid | SELECT /*+ result_cache q_name(Q1) */ COUNT(*) FROM cachejfv c1, cachejfv c2, cachejfv c3, cachejfv c4, cachejfv c5, cachej |
| Result | Published | SELECT /*+ result_cache q_name(Q1) */ COUNT(*) FROM cachejfv c1, cachejfv c2, cachejfv c3, cachejfv c4, cachejfv c5, cachej |

The 'Script Output' pane at the bottom shows the task completed in 0.034 seconds.

- Again, it takes some time to execute the query. The result cache shows that a new entry has been added for the new result.

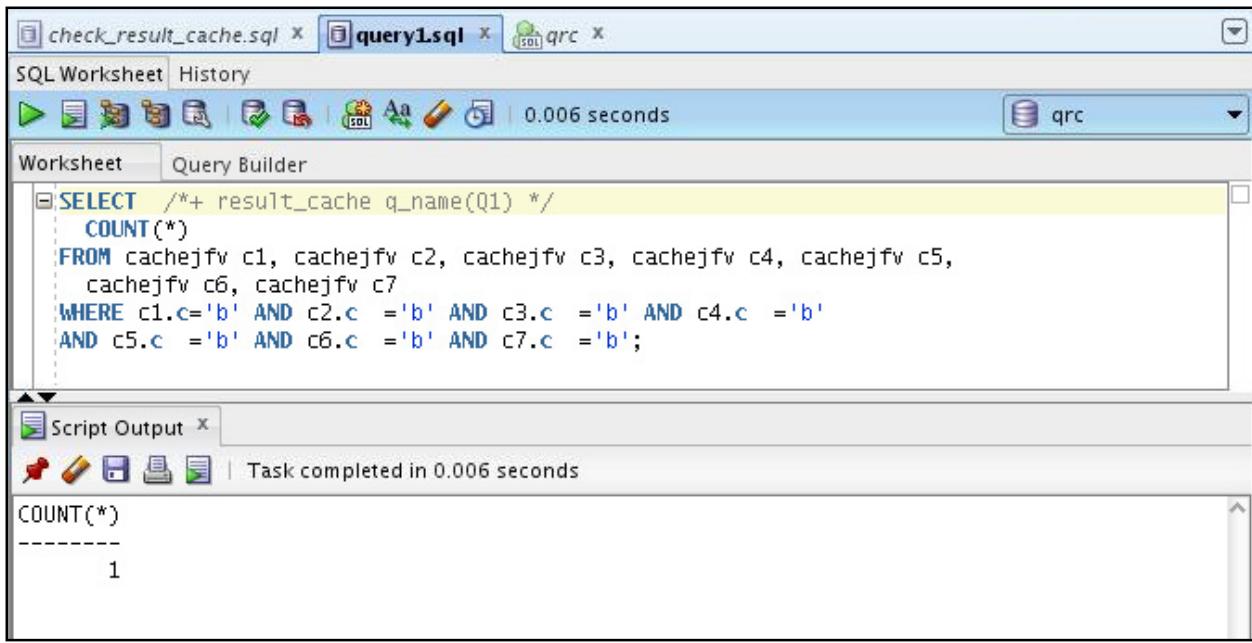
13. Generate a detailed result cache memory report by entering the following commands:

```
set serveroutput on
EXEC DBMS_RESULT_CACHE.MEMORY_REPORT(detailed=>true);
```

The screenshot shows the Oracle SQL Developer interface. In the top tab bar, there are three tabs: 'check_result_cache.sql x', 'query1.sql x', and 'qrc x'. The 'qrc' tab is active. Below the tabs, a toolbar contains icons for running scripts, saving, opening, and other database operations. The main area has two tabs: 'Worksheet' and 'Query Builder', with 'Worksheet' selected. A script is being run in the Worksheet tab:set serveroutput on
EXEC DBMS_RESULT_CACHE.MEMORY_REPORT(detailed=>true);The output window, titled 'Script Output x', shows the results of the executed command. It starts with 'anonymous block completed' and then provides a detailed memory report. The report includes parameters like Block Size (1K bytes), Maximum Cache Size (2080K bytes, 2080 blocks), and Maximum Result Size (104K bytes, 104 blocks). It then breaks down memory usage into categories such as Fixed Memory, Dynamic Memory, Overhead, and various internal components like Hash Table, Chunk Ptrs, and Dependencies.

```
anonymous block completed  
Result Cache Memory Report  
[Parameters]  
Block Size = 1K bytes  
Maximum Cache Size = 2080K bytes (2080 blocks)  
Maximum Result Size = 104K bytes (104 blocks)  
[Memory]  
Total Memory = 120236 bytes [0.036% of the Shared Pool]  
  
... Fixed Memory = 9440 bytes [0.003% of the Shared Pool]  
..... Memory Mgr = 124 bytes  
..... Bloom Fltr = 2K bytes  
..... Cache Mgr = 4416 bytes  
  
..... State Objs = 2852 bytes  
... Dynamic Memory = 110796 bytes [0.033% of the Shared Pool]  
..... Overhead = 78028 bytes  
..... Hash Table = 32K bytes (4K buckets)  
..... Chunk Ptrs = 12K bytes (3K slots)  
..... Chunk Maps = 12K bytes  
..... Miscellaneous = 20684 bytes  
..... Cache Memory = 32K bytes (32 blocks)  
..... Unused Memory = 29 blocks  
..... Used Memory = 3 blocks  
..... Dependencies = 1 blocks (1 count)  
..... Results = 2 blocks  
..... SQL = 1 blocks (1 count)  
..... Invalid = 1 blocks (1 count)
```

14. Rerun the query1.sql script. What do you observe?



A screenshot of the Oracle SQL Developer interface. The top menu bar shows three tabs: 'check_result_cache.sql' (selected), 'query1.sql', and 'qrc'. The toolbar includes icons for running queries, saving, and zooming. The main area is a 'Worksheet' tab showing the following SQL code:

```
SELECT /*+ result_cache q_name(Q1) */  
  COUNT(*)  
FROM cachejfv c1, cachejfv c2, cachejfv c3, cachejfv c4, cachejfv c5,  
  cachejfv c6, cachejfv c7  
WHERE c1.c='b' AND c2.c = 'b' AND c3.c = 'b' AND c4.c = 'b'  
AND c5.c = 'b' AND c6.c = 'b' AND c7.c = 'b';
```

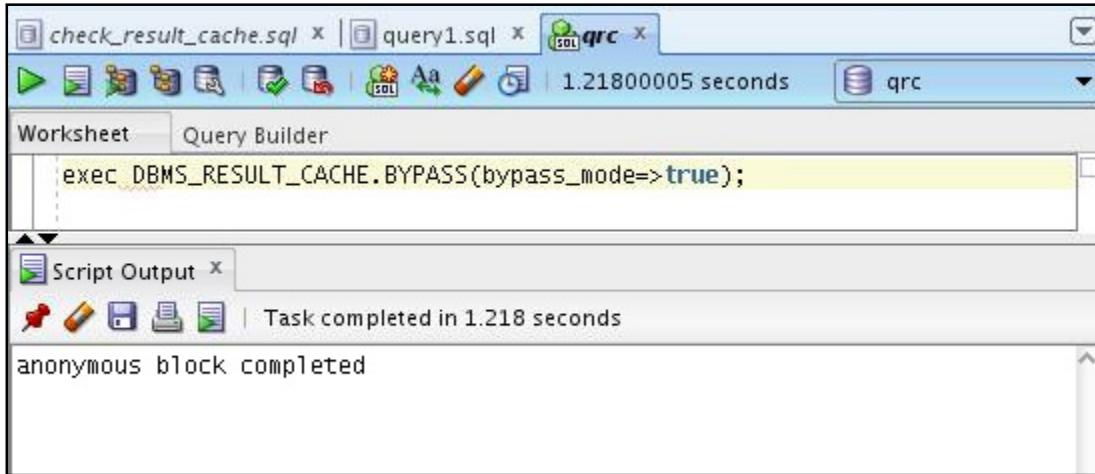
The 'Script Output' tab below shows the results:

```
COUNT(*)  
-----  
1
```

- The query again uses the result that was previously cached.

15. Ensure that you bypass the result cache before performing the next step.

```
exec DBMS_RESULT_CACHE.BYPASS(bypass_mode=>true);
```



A screenshot of the Oracle SQL Developer interface. The top menu bar shows three tabs: 'check_result_cache.sql' (selected), 'query1.sql', and 'qrc'. The toolbar includes icons for running queries, saving, and zooming. The main area is a 'Worksheet' tab showing the following anonymous block:

```
exec DBMS_RESULT_CACHE.BYPASS(bypass_mode=>true);
```

The 'Script Output' tab below shows the results:

```
anonymous block completed
```

16. Rerun query1.sql. What do you observe?

The screenshot shows the Oracle SQL Worksheet interface. The top bar includes tabs for 'SQL Worksheet' and 'History', and various toolbar icons. The status bar at the bottom indicates '1.41400003 seconds'. A connection dropdown shows 'qrc'. The main area is titled 'Worksheet' and contains the following SQL code:

```
SELECT /*+ result_cache q_name(Q1) */  
  COUNT(*)  
FROM cachejfv c1, cachejfv c2, cachejfv c3, cachejfv c4, cachejfv c5,  
  cachejfv c6, cachejfv c7  
WHERE c1.c='b' AND c2.c = 'b' AND c3.c = 'b' AND c4.c = 'b'  
AND c5.c = 'b' AND c6.c = 'b' AND c7.c = 'b';
```

Below the worksheet is a 'Script Output' window with the message 'Task completed in 1.414 seconds'. It displays the output of the query:

```
COUNT(*)  
-----  
1
```

- The query again takes longer to execute because it no longer uses the result cache.
17. Ensure that you no longer bypass the result cache and check whether your query uses it again.
- Execute the following statement:

```
exec DBMS_RESULT_CACHE.BYPASS(bypass_mode=>false);
```

The screenshot shows the Oracle SQL Worksheet interface. The top bar includes tabs for 'check_result_cache.sql' and 'query1.sql', and various toolbar icons. The status bar at the bottom indicates '0.53100002 seconds'. A connection dropdown shows 'qrc'. The main area is titled 'Worksheet' and contains the following SQL code:

```
exec DBMS_RESULT_CACHE.BYPASS(bypass_mode=>false);
```

Below the worksheet is a 'Script Output' window with the message 'Task completed in 0.531 seconds'. It displays the output of the anonymous block:

```
anonymous block completed
```

- Rerun `query1.sql`.

```
SELECT /*+ result_cache q_name(Q1) */  
  COUNT(*)  
FROM cachejfv c1, cachejfv c2, cachejfv c3, cachejfv c4, cachejfv c5,  
  cachejfv c6, cachejfv c7  
WHERE c1.c='b' AND c2.c  ='b' AND c3.c  ='b' AND c4.c  ='b'  
AND c5.c  ='b' AND c6.c  ='b' AND c7.c  ='b';
```

Script Output | Task completed in 0.008 seconds

```
COUNT(*)  
-----  
1
```

18. Open and execute the `query2.sql` script or enter the following query:

```
SELECT COUNT(*)  
FROM cachejfv c1, cachejfv c2, cachejfv c3, cachejfv c4,  
  cachejfv c5, cachejfv c6, cachejfv c7  
WHERE c1.c='b' AND c2.c  ='b' AND c3.c  ='b' AND c4.c  ='b'  
AND c5.c  ='b' AND c6.c  ='b' AND c7.c  ='b';
```

What do you observe?

```
check_result_cache.sql x | query1.sql x | query2.sql x | qrc x
```

SQL Worksheet History

1.39600003 seconds

```
Worksheet Query Builder
```

```
SELECT COUNT(*)  
FROM cachejfv c1, cachejfv c2, cachejfv c3, cachejfv c4,  
  cachejfv c5, cachejfv c6, cachejfv c7  
WHERE c1.c='b' AND c2.c  ='b' AND c3.c  ='b' AND c4.c  ='b'  
AND c5.c  ='b' AND c6.c  ='b' AND c7.c  ='b';
```

Script Output | Task completed in 1.396 seconds

```
COUNT(*)  
-----  
1
```

- Although the query is the same as the one in query1.sql, it is not recognized as cached because it does not contain the hint. So its execution time is long again.
19. How would you force the previous query to use the cached result without using hints?
- Use the following code to force the use of the cached result or open and execute the force_result_cache.sql script.

```
set echo on

show parameter result_cache_mode

select type,status,name,object_no,row_count,row_size_avg from
v$result_cache_objects order by 1;

alter session set result_cache_mode=force;
```

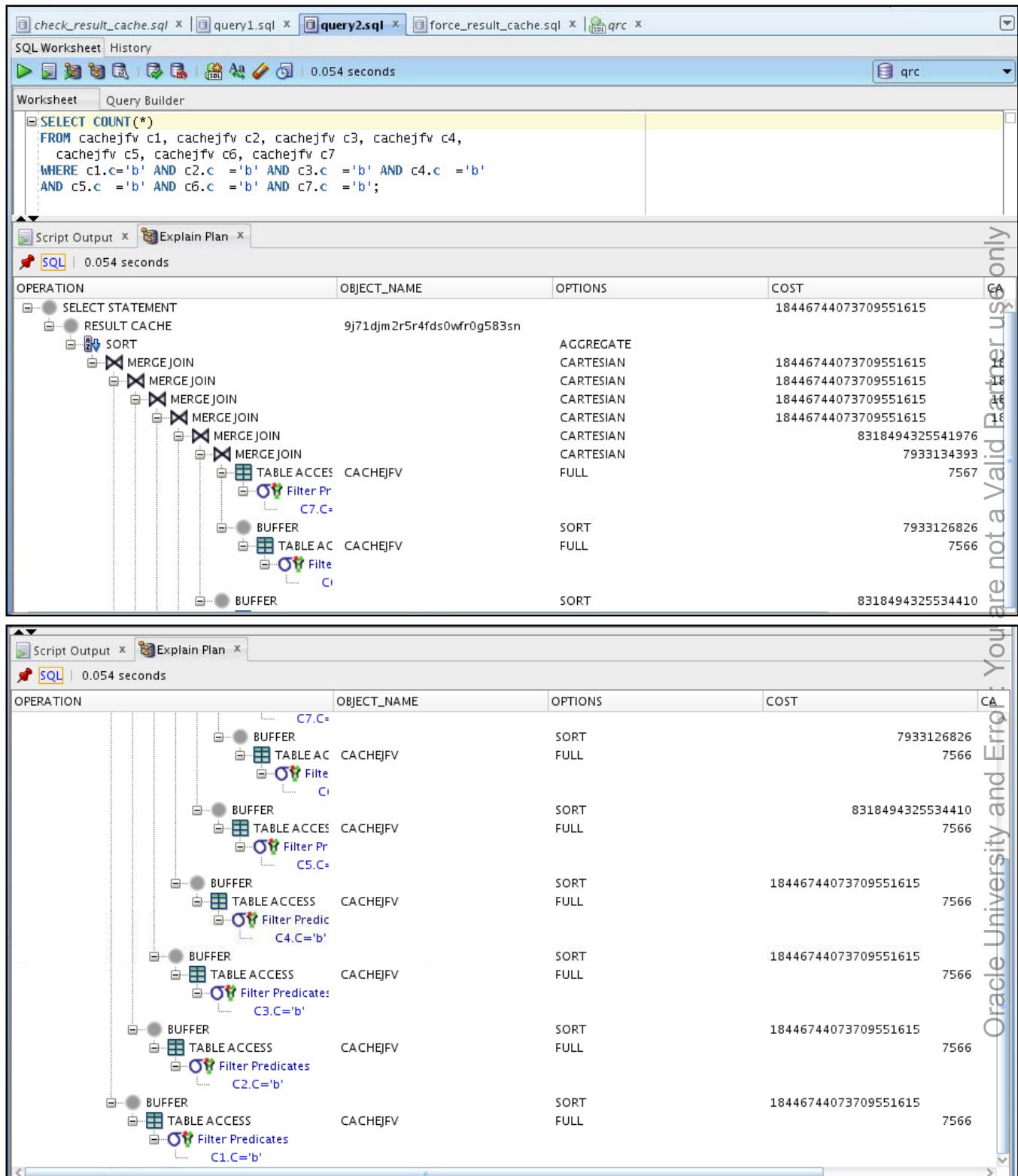
The screenshot shows the Oracle SQL Worksheet interface. The title bar has tabs for check_result_cache.sql, query1.sql, query2.sql, force_result_cache.sql (which is currently selected), and qrc. The toolbar includes icons for New, Open, Save, Print, Copy, Paste, Find, and Execute. The status bar shows "0.011 seconds". The main area is a "Worksheet" tab with the following SQL code:

```
set echo on
show parameter result_cache_mode
select type,status,name,object_no,row_count,row_size_avg from v$result_cache_objects order by 1;
alter session set result_cache_mode=force;
```

Below the code, the "Script Output" window displays the results of the execution:

```
> show parameter result_cache_mode
NAME          TYPE    VALUE
result_cache_mode      string  MANUAL
> select type,status,name,object_no,row_count,row_size_avg from v$result_cache_objects order by 1
TYPE      STATUS     NAME
-----  -----  -----
Dependency Published QRC.CACHEJFV
Result     Invalid   SELECT /*+ result_cache q_name(Q1) */
               COUNT(*)
               FROM cachejfv c1, cachejfv c2, cachejfv c3, cachejfv c4, cachejfv c5,
               cachej
Result     Published  SELECT /*+ result_cache q_name(Q1) */
               COUNT(*)
               FROM cachejfv c1, cachejfv c2, cachejfv c3, cachejfv c4, cachejfv c5,
               cachej
> alter session set result_cache_mode=force
session SET altered.
```

- View the explain plan for the query2.sql script.



- e. Rerun the `query2.sql` script to verify that the query runs instantaneously because you successfully used the cached result.

A screenshot of the Oracle SQL Developer interface. The title bar shows multiple tabs: check_result_cache.sql, query1.sql, query2.sql (which is selected), and force_result_cache.sql. The status bar at the bottom indicates "0.023 seconds". The main area is a Worksheet tab showing the following SQL query:

```
SELECT COUNT(*)  
FROM cachejfv c1, cachejfv c2, cachejfv c3, cachejfv c4,  
    cachejfv c5, cachejfv c6, cachejfv c7  
WHERE c1.c='b' AND c2.c = 'b' AND c3.c = 'b' AND c4.c = 'b'  
AND c5.c = 'b' AND c6.c = 'b' AND c7.c = 'b';
```

The output window below shows the result of the query:

```
COUNT(*)  
-----  
1
```

f. Finally, undo your change.

```
alter session set result_cache_mode=manual;
```

A screenshot of the Oracle SQL Developer interface. The title bar shows multiple tabs: check_result_cache.sql, query1.sql, query2.sql, and force_result_cache.sql. The status bar at the bottom indicates "0.002 seconds". The main area is a Worksheet tab showing the following SQL command:

```
alter session set result_cache_mode=manual;
```

The output window below shows the result of the command:

```
session SET altered.
```

20. Clear the result cache. Query V\$RESULT_CACHE_OBJECTS to verify the clear operation.

```
exec dbms_result_cache.flush;
```

A screenshot of the Oracle SQL Developer interface. The title bar shows multiple tabs: check_result_cache.sql, query1.sql, query2.sql, and force_result_cache.sql. The status bar at the bottom indicates "0.30199999 seconds". The main area is a Worksheet tab showing the following PL/SQL command:

```
exec dbms_result_cache.flush;
```

The output window below shows the result of the command:

```
anonymous block completed
```

21. Open and execute the cre_func.sql script. This script creates a PL/SQL function that uses the result cache.

The screenshot shows the Oracle SQL Worksheet interface. In the top tab bar, there are four tabs: query1.sql, query2.sql, cre_func.sql (which is selected), and qrc. The main workspace contains the following PL/SQL code:

```

create or replace function CACHEJFV_COUNT(v varchar2)
return number
result_cache relies_on (cachejfv)
is
  cnt number;
begin
  select count(*) into cnt
  from cachejfv c1,cachejfv c2,cachejfv c3,cachejfv c4,cachejfv c5,cachejfv c6, cachejfv c7
  where c1.c=v and c2.c=v and c3.c=v and c4.c=v and c5.c=v and c6.c=v and c7.c=v;
  return cnt;
end;
/

```

Below the workspace is a Script Output window showing the result of the compilation:

```

FUNCTION CACHEJFV_COUNT compiled

```

22. Determine what is in the result cache by querying V\$RESULT_CACHE_OBJECTS. (This is the check_result_cache.sql script.)

```

select type,status,name,object_no,row_count,row_size_avg
from v$result_cache_objects
order by 1;

```

The screenshot shows the Oracle SQL Worksheet interface. In the top tab bar, there are four tabs: query1.sql, query2.sql, cre_func.sql, and check_result_cache.sql (which is selected). The main workspace contains the same SQL query as the previous screenshot:

```

select type,status,name,object_no,row_count,row_size_avg
from v$result_cache_objects
order by 1;

```

Below the workspace is a Script Output window showing the result of the query:

| TYPE | STATUS | NAME | OBJECT... | ROW_C... | ROW_SI... |
|------|--------|------|-----------|----------|-----------|
| | | | | | |

The output indicates "All Rows Fetched: 0 in 0.001 seconds".

23. Call the new function with 'b' as its argument. What do you observe?

```

select cachejfv_count('b') from dual;

```

A screenshot of the Oracle SQL Developer interface. The top menu bar shows several open tabs: query1.sql, query2.sql, cre_func.sql, check_result_cache.sql, and qrc. The status bar at the bottom of the menu bar displays "3.23300004 seconds". The main area has two tabs: Worksheet and Query Builder, with Worksheet selected. A script window contains the SQL command: "select cachejfv_count('b') from dual;". Below the script is a "Script Output" window titled "Script Output X". It shows the message "Task completed in 3.233 seconds" followed by the result of the function call: "CACHEJFV_COUNT('B')". The output is a single line with the value "1".

- It takes a long time to execute because the result is not cached yet. After executing the function, the function's result for the 'b' argument is cached.

24. Call the new function with 'b' as its argument again. What do you observe?

```
select cachejfv_count('b') from dual;
```

A screenshot of the Oracle SQL Developer interface, identical to the previous one except for the execution time. The status bar at the bottom of the menu bar now displays "0.003 seconds". The "Script Output" window shows the message "Task completed in 0.003 seconds" followed by the result of the function call: "CACHEJFV_COUNT('B')". The output is a single line with the value "1".

- This time the function executes almost instantaneously.

25. Call the new function again, but with 'c' as its argument. What do you observe?

The screenshot shows the Oracle SQL Developer interface. In the top navigation bar, there are several tabs: query1.sql, query2.sql, cre_func.sql, check_result_cache.sql, and qrc. The qrc tab is currently active. A red box highlights the status bar at the bottom of the window, which displays "3.14299989 seconds". Below the status bar, the "Worksheet" tab is selected, showing the SQL command: "select cachejfv_count('c') from dual;". In the "Script Output" pane, the results are displayed: "Task completed in 3.143 seconds" followed by the output "CACHEJFV_COUNT('C')-----1".

- Again, it takes a long time to execute the function because of the new value for the argument. After execution, the second result is cached.

Practice 10-2: Using Other Access Paths (Optional)

Overview

In this practice, you explore the various access paths that the optimizer can use, and compare them. You have the possibility of exploring three scenarios, each of which is self-contained. All scripts needed for this practice can be found in your \$HOME/labs/solutions/Access_Paths directory.

Tasks

1. Open the connection named `sh_connection` that you created in Practice 2. If you have not created this connection, create it now with the following attributes:

- a. Click the New Connection button.

- b. Create a connection with the following specifications:

Name: `sh_connection`

Username: `sh`

Password: `sh`

Select “Save Password.”

Sid: `systun`

Click Test.

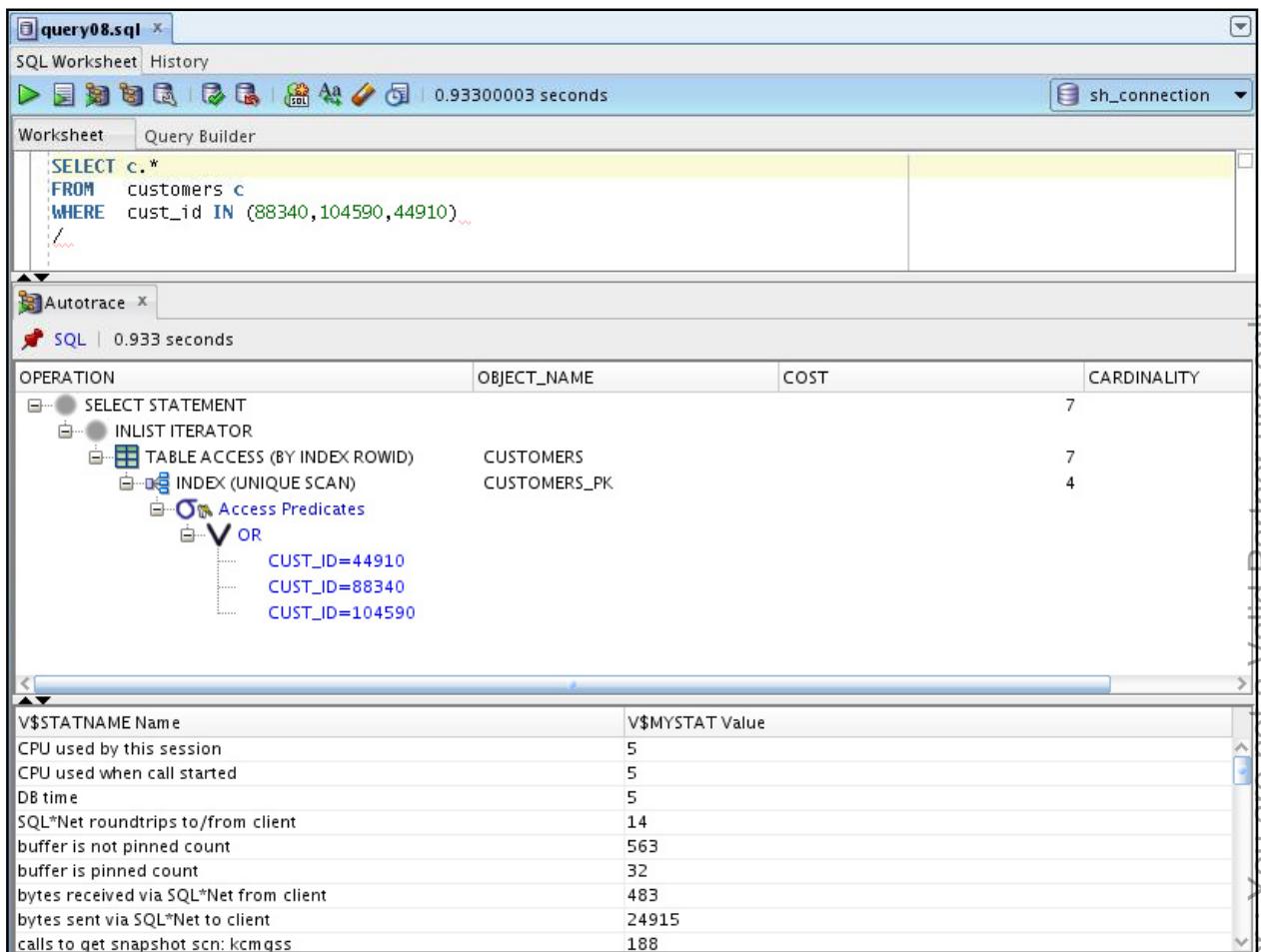
Click Connect.

2. **Case 1: Inlist Iterator:** Open the

\$HOME/labs/solutions/Access_Paths/query08.sql script. Use `sh_connection` to autotrace the query.

```
SELECT c.*  
FROM   customers c  
WHERE  cust_id IN (88340,104590,44910);
```

What do you observe?



- The optimizer can use the CUSTOMERS primary key index to resolve this query.
The cost is very low for the resulting plan.

3. **Case 2: Using Hash Clusters:** The SHC schema was created and populated in the practice setup using \$HOME/labs/solutions/Access_Paths/shc_setup.sql to set up your environment for this practice. This script creates the bigemp_fact table as a member of bigemp_cluster. bigemp_cluster is a single-table hash cluster. The rows of the table are sorted on the deptno and sal columns before the hash function is applied. The script is listed as follows:

```
-- run with sqlplus /nolog @shc_setup.sql
connect / as sysdba

drop user shc cascade;

create user shc identified by shc;
Grant DBA to SHC;
GRANT select_catalog_role to SHC;
GRANT select any dictionary to SHC;
```

```
connect shc/shc

set echo on

set linesize 200

drop cluster bigemp_cluster including tables;

CREATE CLUSTER bigemp_cluster
(deptno number, sal number sort)
HASHKEYS 10000
single table HASH IS deptno SIZE 50
tablespace users;

create table bigemp_fact (
    empno number primary key, sal number sort, job varchar2(12) not
null,
    deptno number not null, hiredate date not null)
CLUSTER bigemp_cluster (deptno, sal);

begin
for i in 1..1400000 loop
    insert into bigemp_fact values(i,i,'J1',10,sysdate);
end loop;
commit;
end;
/

begin
for i in 1..1400000 loop
    insert into bigemp_fact values(1400000+i,i,'J1',20,sysdate);
end loop;
commit;
end;
/

exec dbms_stats.gather_schema_stats('SH');

exit
```

4. Create a connection for the `shc` user. Use this connection for all the scripts in the Hash Cluster case.
 - a. Click the New Connection button.
 - b. Create a connection with the following specifications:

Name: shc
Username: shc
Password: shc
Select "Save Password."
Sid: systun
Click Test.
Click Connect.
5. Use the `shc` connection to execute the `query15a.sql` script. This script sets `workarea_size_policy` to `MANUAL`, and `sort_area_size` to a small value. Because you may have a lot of memory on your system, the script reduces the amount of memory available to your session. It then flushes the cache and shared pool to eliminate the possibility of the cost being reduced by previous queries loading the cache.

The screenshot shows the Oracle SQL Developer interface. At the top, there are three tabs: `query08.sql`, `shc`, and `query15a.sql`. The `query15a.sql` tab is active. Below the tabs is a toolbar with various icons. The main area is divided into two panes: `Worksheet` (top) and `Script Output` (bottom). In the `Worksheet` pane, the following SQL code is visible:set echo off
alter session set workarea_size_policy=manual;
alter session set sort_area_size=50000;

alter system flush shared_pool;
alter system flush buffer_cache;In the `Script Output` pane, the output of the executed script is shown:session SET altered.
session SET altered.
system FLUSH altered.
system FLUSH altered.

6. Use the `shc` connection to autotrace the query in the `query15b.sql` script. What do you observe?

SQL Worksheet History

Worksheet Query Builder

```
select * from bigemp_fact where deptno=10;
```

Autotrace SQL | 0.957 seconds

| OPERATION | OBJECT_NAME | COST | CARDINALITY |
|---------------------|-------------|------|-------------|
| SELECT STATEMENT | | 1 | 1 |
| TABLE ACCESS (HASH) | BIGEMP_FACT | 1 | 1 |
| Access Predicates | DEPTNO=10 | | |

V\$STATNAME Name V\$MYSTAT Value

| | |
|--|--------|
| CPU used by this session | 4 |
| CPU used when call started | 8 |
| DB time | 17 |
| SQL*Net roundtrips to/from client | 14 |
| buffer is not pinned count | 124 |
| buffer is pinned count | 6 |
| bytes received via SQL*Net from client | 438 |
| bytes sent via SQL*Net to client | 24600 |
| calls to get snapshot scn: kcmgss | 39 |
| calls to kcmgcs | 2 |
| cell physical IO interconnect bytes | 278528 |
| cluster key scan block gets | 11 |

- The optimizer decides to use the cluster access path to retrieve data. The cost is minimal.

7. Execute the query15a.sql script again.

SQL Worksheet History

Worksheet Query Builder

```
set echo off
alter session set workarea_size_policy=manual;
alter session set sort_area_size=50000;

alter system flush shared_pool;
alter system flush buffer_cache;
```

Script Output Task completed in 0.142 seconds

```
session SET altered.
session SET altered.
system FLUSH altered.
system FLUSH altered.
```

8. Open and autotrace the query in the `query16.sql` script as the `SHC` user. What do you observe?

The screenshot shows the Oracle SQL Developer interface. In the top navigation bar, the tab `query16.sql` is selected. Below the tabs, the status bar indicates `1.75100005 seconds`. The main area contains a worksheet with the following SQL query:

```
select * from bigemp_fact where deptno=10 order by sal;
```

Below the worksheet is the `Autotrace` window. It displays the execution plan for the query. The plan is as follows:

- SELECT STATEMENT
- TABLE ACCESS (HASH)
- Access Predicates
 DEPTNO=10

A red rectangle highlights the Access Predicates section of the execution plan. At the bottom of the `Autotrace` window, there is a table titled `V$STATNAME` with the following data:

| V\$STATNAME | Name | V\$MYSTAT | Value |
|--|------|-----------|-------|
| CPU used by this session | | 3 | |
| CPU used when call started | | 7 | |
| DB time | | 6 | |
| SQL*Net roundtrips to/from client | | 14 | |
| buffer is not pinned count | | 182 | |
| buffer is pinned count | | 6 | |
| bytes received via SQL*Net from client | | 451 | |
| bytes sent via SQL*Net to client | | 24684 | |
| calls to get snapshot scn: kcmgss | | 59 | |
| calls to kcm gcs | | 2 | |
| cell physical IO interconnect bytes | | 327680 | |

- The script executes a slightly different query, one that requires ordering the result based on the sorted `sal` column. The optimizer can still use the cluster access path without sorting the data. The cost is still minimal.

9. Execute the `query15a.sql` script again.

The screenshot shows the Oracle SQL Developer interface. The top tab bar has tabs for 'query08.sql', 'shc', 'query15a.sql' (which is selected), 'query16.sql', and 'query15b.sql'. The main workspace is a 'Worksheet' tab showing the following SQL code:

```
set echo off
alter session set workarea_size_policy=manual;
alter session set sort_area_size=50000;

alter system flush shared_pool;
alter system flush buffer_cache;
```

Below the worksheet is a 'Script Output' window showing the results of the executed commands:

```
session SET altered.
session SET altered.
system FLUSH altered.
system FLUSH altered.
```

10. Autotrace the query in the query17.sql script. What do you observe?

The screenshot shows the Oracle SQL Developer interface with the 'query17.sql' script selected. The main workspace shows the query:

```
select * from bigemp_fact where deptno=10 order by sal desc;
```

Below the workspace is an 'Autotrace' window showing the execution plan:

| OPERATION | OBJECT_NAME | OPTIONS | COST |
|-------------------|-------------|---------|------|
| SELECT STATEMENT | | | 1 |
| TABLE ACCESS | BIGEMP_FACT | HASH | 1 |
| Access Predicates | | | |
| DEPTNO=10 | | | |

- The query17.sql script executes the query; the difference is that the result is ordered based on the sorted sal column in descending order. The optimizer can still use the cluster access path without sorting the data. The cost is still minimal.

11. Execute the query15a.sql script again.

```

set echo off
alter session set workarea_size_policy=manual;
alter session set sort_area_size=50000;

alter system flush shared_pool;
alter system flush buffer_cache;

```

Script Output

Task completed in 0.123 seconds

session SET altered.
session SET altered.
system FLUSH altered.
system FLUSH altered.

12. Open and autotrace the query in the `query18.sql` script as the SHC user. What do you observe?

Auttrace X

SQL | 8.405 seconds

| OPERATION | OBJECT_NAME | COST | CARDINALITY |
|---------------------|-------------|-------|-------------|
| SELECT STATEMENT | | 48063 | |
| SORT (ORDER BY) | | 48063 | 1400000 |
| TABLE ACCESS (HASH) | BIGEMP_FACT | 1 | 1400000 |
| Access Predicates | | | |
| DEPTNO=10 | | | |

V\$STATNAME Name V\$MYSTAT Value

| | |
|-----------------------------------|-----|
| CPU used by this session | 392 |
| CPU used when call started | 395 |
| DB time | 751 |
| SQL*Net roundtrips to/from client | 14 |
| buffer is not pinned count | 124 |
| buffer is pinned count | 6 |

- The script executes the same query, but this time asks to order the result based on the nonsorted `empno` column. The optimizer can still make use of the cluster access path, but must sort the data making the cost of the query higher.

13. Execute the `query15a.sql` script again.

```

set echo off
alter session set workarea_size_policy=manual;
alter session set sort_area_size=50000;

alter system flush shared_pool;
alter system flush buffer_cache;

```

session SET altered.
session SET altered.
system FLUSH altered.
system FLUSH altered.

14. Autotrace the query in the query19.sql script. What do you observe?

```

select * from bigemp_fact where deptno=10 order by sal,empno;

```

| OPERATION | OBJECT_NAME | COST | CARDINALITY |
|---------------------|-------------|-------|-------------|
| SELECT STATEMENT | | 48063 | 1400000 |
| SORT (ORDER BY) | | 48063 | 1400000 |
| TABLE ACCESS (HASH) | BIGEMP_FACT | 1 | 1400000 |
| Access Predicates | | | |
| DEPTNO=10 | | | |

| V\$STATNAME | Name | V\$MYSTAT | Value |
|--|------|-----------|-------|
| CPU used by this session | | 305 | |
| CPU used when call started | | 310 | |
| DB time | | 366 | |
| SQL*Net roundtrips to/from client | | 14 | |
| buffer is not pinned count | | 126 | |
| buffer is pinned count | | 6 | |
| bytes received via SQL*Net from client | | 457 | |
| bytes sent via SQL*Net to client | | 24730 | |
| calls to get snapshot scn: kcmgss | | 40 | |
| calls to kcmgcs | | 2 | |

- The script executes the same query, but this time asks to order the result based on the sal, empno key. The optimizer can make use of the cluster access path, but must sort the data making the cost of the query higher.

15. **Case 3: Using Index Cluster:** The nic_setup.sql script was executed as part of the setup for this practice. This script creates and populates the nic schema to set up your

environment for this case. It creates large emp and dept tables. Each of these tables has an index on the deptno value. The nic_setup.sql script is listed as follows:

```
-- run with sqlplus /nolog
connect / as sysdba

DROP user NIC cascade;

create user nic identified by nic;
GRANT DBA to NIC;
GRANT SELECT_CATALOG_ROLE to NIC;
GRANT select any dictionary to NIC;

connect nic/nic

set echo on

drop cluster emp_dept including tables;

drop table emp purge;
drop table dept purge;

CREATE TABLE emp (
    empno      NUMBER(7)          ,
    ename      VARCHAR2(15) NOT NULL,
    job        VARCHAR2(9)         ,
    mgr        NUMBER(7)          ,
    hiredate   DATE              ,
    sal        NUMBER(7)          ,
    comm       NUMBER(7)          ,
    deptno    NUMBER(3)          )
;

CREATE TABLE dept (
    deptno    NUMBER(3)          ,
    dname     VARCHAR2(14)        ,
    loc       VARCHAR2(14)        ,
    c         VARCHAR2(500)
)
;

CREATE INDEX emp_index
ON emp(deptno)
TABLESPACE users
STORAGE (INITIAL 50K
NEXT 50K
MINEXTENTS 2
```

```
MAXEXTENTS 10
PCTINCREASE 33);

CREATE INDEX dept_index
  ON dept(deptno)
  TABLESPACE users
  STORAGE (INITIAL 50K
    NEXT 50K
    MINEXTENTS 2
    MAXEXTENTS 10
    PCTINCREASE 33);

begin
  for i in 1..999 loop
    insert into dept values
    (i,'D'||i,'L'||i,dbms_random.string('u',500));
  end loop;
  commit;
end;
/

begin
  for i in 1..500000 loop
    insert into emp values
    (i,dbms_random.string('u',15),dbms_random.string('u',9),i,sysdate,i,i,
    mod(i,999));
  end loop;
  commit;
end;
/

exec dbms_stats.gather_schema_stats('SH');

exit;
```

16. Create a connection for the `nic` user.
 - a. Click the New Connection button.
 - b. Create a connection with the following specifications:
Name: nic
Username: nic
Password: nic
Select “Save Password.”
Sid: systun

Click Test.

Click Connect.

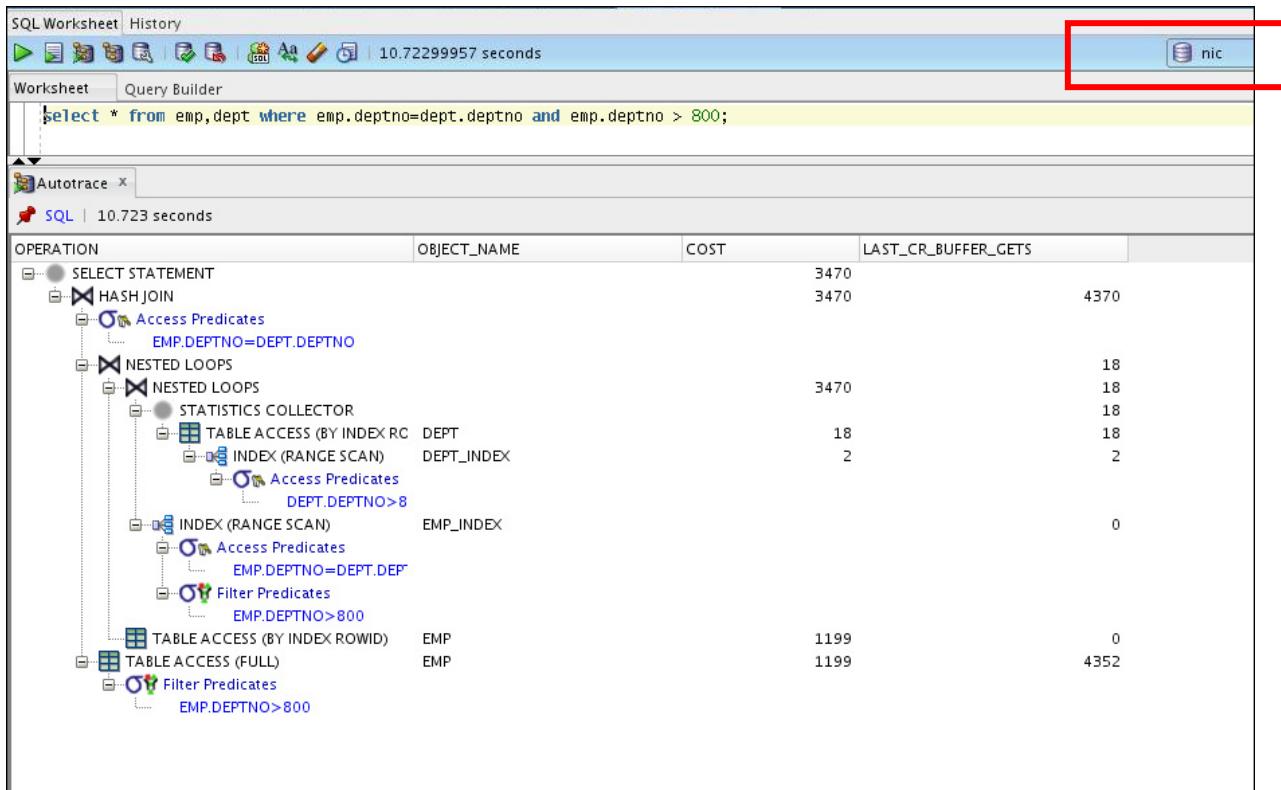
17. Use the nic connection to execute the nic_query_a.sql script.

Note: sort_area_size remains small and hash_area_size is also set to a small value.

```
SQL Worksheet History
Worksheet Query Builder
ALTER SESSION SET WORKAREA_SIZE_POLICY=MANUAL;
ALTER SESSION SET SORT_AREA_SIZE=50000;
ALTER SESSION SET HASH_AREA_SIZE=5000;

Script Output
Task completed in 0.003 seconds
session SET altered.
session SET altered.
session SET altered.
```

18. Open and autotrace the query in the nic_query_b.sql script as the NIC user. What do you observe?



- The script executes a join between the EMP and DEPT tables. The optimizer is able to make use of the index to resolve the join.

19. How would you enhance the performance of the previous query? Implement your solution.

- a. The `ic` user was created in the setup of the practices with the `ic_setup.sql` script to create an index cluster to store the two tables. These tables have exactly the same rows in both the `nic` and `ic` schemas. This script creates an index cluster containing the `emp` and `dept` tables. The rows of these tables are stored together clustered by the `deptno` value. This script is listed as follows:

```
-- run with sqlplus /nolog

connect / as sysdba

DROP user IC cascade;

CREATE USER ic IDENTIFIED BY ic;

GRANT DBA TO ic;
GRANT SELECT_CATALOG_ROLE TO ic;
GRANT SELET ANY DICTIONARY TO ic;

connect ic/ic

set echo on

drop table emp purge;
drop table dept purge;

drop cluster emp_dept including tables;

CREATE CLUSTER emp_dept (deptno NUMBER(3))
  SIZE 600
  TABLESPACE users
  STORAGE (INITIAL 200K
    NEXT 300K
    MINEXTENTS 2
    PCTINCREASE 33);

CREATE TABLE emp (
  empno      NUMBER(7)          ,
  ename      VARCHAR2(15) NOT NULL,
  job        VARCHAR2(9)         ,
  mgr        NUMBER(7)          ,
  hiredate   DATE              ,
  sal        NUMBER(7)          ,
  comm       NUMBER(7)          ,
  deptno    NUMBER(3)) CLUSTER emp_dept (deptno);
```

```
CREATE TABLE dept (
    deptno NUMBER(3) ,
    dname  VARCHAR2(14) ,
    loc    VARCHAR2(14) ,
    c      VARCHAR2(500))
CLUSTER emp_dept (deptno);

CREATE INDEX emp_dept_index
    ON CLUSTER emp_dept
    TABLESPACE users
    STORAGE (INITIAL 50K
        NEXT 50K
        MINEXTENTS 2
        MAXEXTENTS 10
        PCTINCREASE 33);

begin
    for i in 1..999 loop
        insert into dept values
(i,'D'||i,'L'||i,dbms_random.string('u',500));
    end loop;
    commit;
end;
/

begin
    for i in 1..500000 loop
        insert into emp values
(i,dbms_random.string('u',15),dbms_random.string('u',9),i,sysdate,i,i,
mod(i,999));
    end loop;
    commit;
end;
/

exec dbms_stats.gather_schema_stats('SH');

exit;
```

20. Create a connection for the `ic` user.
 - a. Click the New Connection button.
 - b. Create a connection with the following specifications:
Name: `ic`
Username: `ic`

Password: ic
Select “Save Password.”
Sid: systun
Click Test.
Click Connect.

21. Open and execute the `nic_query_a.sql` script as the `ic` user.

The screenshot shows the Oracle SQL Worksheet interface. The title bar has tabs for shc, nic, nic_query_a.sql (which is active), nic_query_b.sql, and ic. The main area is titled "Worksheet" and contains the following SQL code:

```
alter session set workarea_size_policy=manual;
alter session set sort_area_size=50000;
alter session set hash_area_size=5000;
```

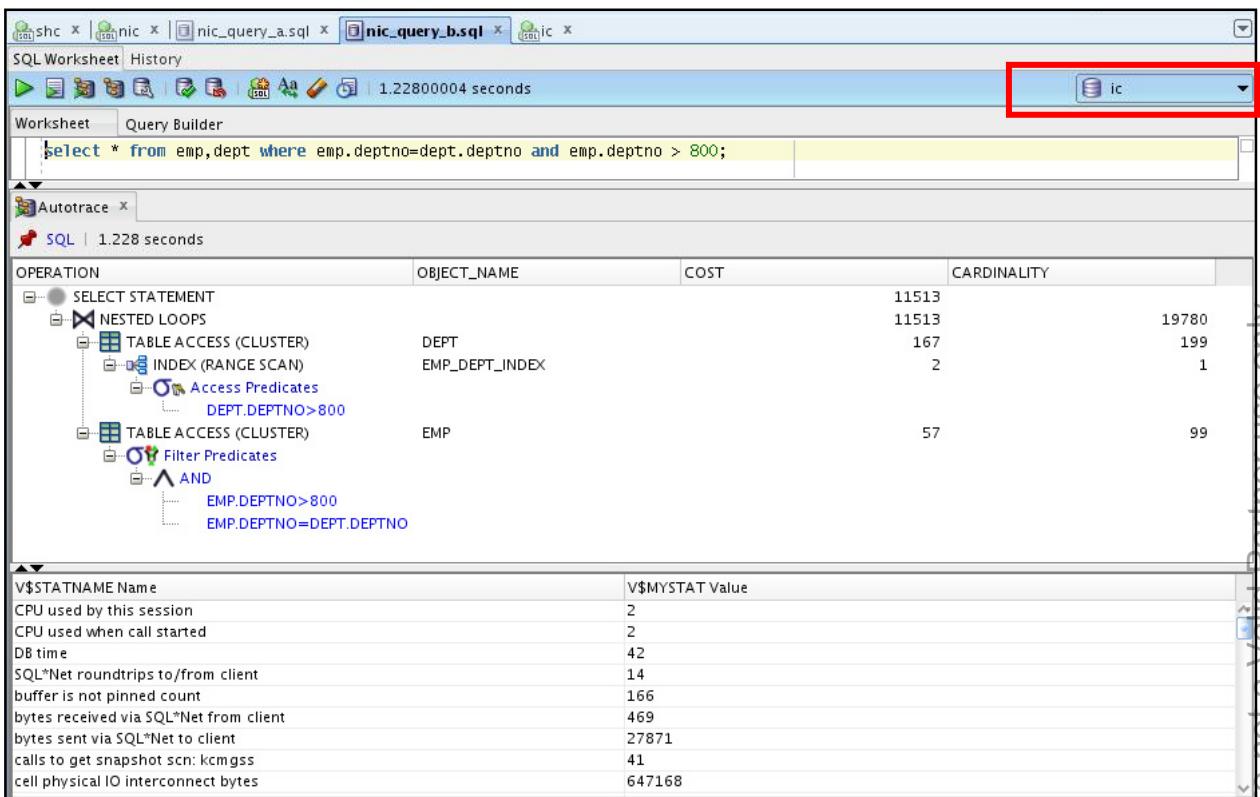
Below the worksheet is a "Script Output" window showing the results of the execution:

```
session SET altered.
session SET altered.
session SET altered.
```

22. Use the `ic` connection to autotrace the query in the `nic_query_b.sql` script again.

```
select *
  from emp,dept
 where emp.deptno=dept.deptno and emp.deptno > 800;
```

What do you observe?



- The optimizer is able to use the cluster access path that makes the query execute faster.

- Important: Close all connections to the SH user. In SQL Developer, find and close all sh_connection windows. Select sh_connection in the Connections pane and disconnect. Close all SQL*Plus sessions.
- Execute the ap_cleanup.sh script to clean up your environment for this practice. This script rebuilds the SH schema. If the User dropped message does not appear, or there is an error message in place of it, wait until the script finishes, find and exit from any sessions that are connected as the SH user, and then execute this script again.

Note: Please ignore the OLAP errors.

```
$ ./ap_cleanup.sh
...
Connected to:...

SQL>
SQL> @sh_main.sh example temp oracle_4U
/u01/app/oracle/product/12.1.0/dbhome_1/demo/schema/sales_history/ /home/oracle/ v3
SQL> Rem
SQL> Rem $Header: sh_main.sql 06-mar-2008.15:00:45 cbauwens Exp
$#
SQL> Rem
```

```
SQL> Rem sh_main.sql
SQL> Rem
SQL> Rem Copyright (c) 2001, 2008, Oracle. All rights reserved.
SQL> Rem
SQL> Rem      NAME
SQL> Rem          sh_main.sql - Main schema creation and load
script
SQL> Rem
SQL> Rem      DESCRIPTION
SQL> Rem          SH is the Sales History schema of the Oracle
Sample
SQL> Rem          Schemas
SQL> Rem
SQL> Rem      NOTES
SQL> Rem          CAUTION: use absolute pathnames as parameters 5
and 6.
SQL> Rem          Example (UNIX) echo
$ORACLE_HOME/demo/schema/sales_history
SQL> Rem          Please make sure that parameters 5 and 6 are
specified
SQL> Rem          including the trailing directory delimiter,
since the
SQL> Rem          directory parameters and the filenames are
concatenated
SQL> Rem          without adding any delimiters.
SQL> Rem          Run this as SYS or SYSTEM
SQL> Rem
SQL> Rem      MODIFIED (MM/DD/YY)
SQL> Rem          cbauwens    03/06/08 - NLS settings for load
SQL> Rem          cbauwens    07/10/07 - NLS fix bug 5684394
SQL> Rem          glyon      06/28/07 - grant CWM_USER role, if it
exists
SQL> Rem          cbauwens    02/23/05 - deprecating connect
role
SQL> Rem          ahunold    10/14/02 -
> Rem      hyeh     08/29/02 - hyeh_mv_comschema_to_rdbms
SQL> Rem          ahunold    08/20/02 - path > dir
SQL> Rem          ahunold    08/15/02 - versioning
SQL> Rem          ahunold    04/30/02 - Reduced DIRECTORY privileges
SQL> Rem          ahunold    08/28/01 - roles
SQL> Rem          ahunold    07/13/01 - NLS Territory
SQL> Rem          ahunold    04/13/01 - spool, notes
SQL> Rem          ahunold    04/10/01 - flexible log and data paths
SQL> Rem          ahunold    03/28/01 - spool
```

```
SQL> Rem ahunold 03/23/01 - absolute path names
SQL> Rem ahunold 03/14/01 - prompts
SQL> Rem ahunold 03/09/01 - privileges
SQL> Rem hbaer 03/01/01 - changed loading from COSTS
      table from
SQL> Rem                                     SQL*Loader to external table
      with GROUP BY
SQL> Rem                                         Added also CREATE DIRECTORY
      privilege
SQL> Rem
SQL>
SQL> SET ECHO OFF

specify password for SH as parameter 1:

specify default tablespace for SH as parameter 2:

specify temporary tablespace for SH as parameter 3:

specify password for SYS as parameter 4:

specify directory path for the data files as parameter 5:

writeable directory path for the log files as parameter 6:

specify version as parameter 7:

Session altered.

User dropped.

old 1: CREATE USER sh IDENTIFIED BY &pass
new 1: CREATE USER sh IDENTIFIED BY sh

User created.

old 1: ALTER USER sh DEFAULT TABLESPACE &tbs
new 1: ALTER USER sh DEFAULT TABLESPACE example
old 2: QUOTA UNLIMITED ON &tbs
new 2: QUOTA UNLIMITED ON example

User altered.
```

```
old    1: ALTER USER sh TEMPORARY TABLESPACE &ttbs
new    1: ALTER USER sh TEMPORARY TABLESPACE temp

User altered.

Grant succeeded.

...
<<<< FINAL PROCESSING >>>>
- Changes have been committed

PL/SQL procedure successfully completed.

Commit complete.

gathering statistics ...

PL/SQL procedure successfully completed.

PL/SQL procedure successfully completed.

Disconnected ...
$
```

Practices for Lesson 11: Introduction to Optimizer Statistics Concepts

Chapter 11

Practices for Lesson 11: Overview

Practices Overview

In these practices, you perform the following:

- Experiment with the index clustering factor
- Create expression statistics
- Enable automatic statistics gathering(Optional)
- Use system statistics(Optional)

Practice 11-1: Index Clustering Factor

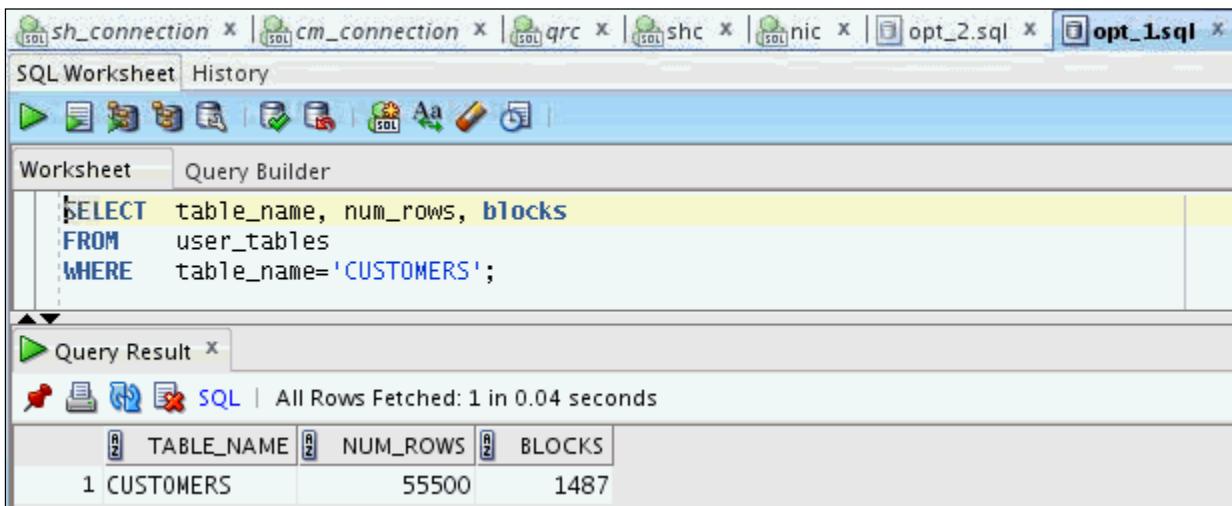
Overview

In this practice, you experiment with the index clustering factor. Here, you learn how the optimizer uses the index clustering factor to determine whether using an index is more effective than a full table scan.

Tasks

1. Change directories to \$HOME/labs/solutions/Optimizer_Statistics. Start a SQL Developer session, connect to the database as SH user, and then query the number of rows and blocks in the sh.customers table. Use the opt_1.sql file.

```
SELECT    table_name, num_rows, blocks
FROM      user_tables
WHERE     table_name='CUSTOMERS';
```



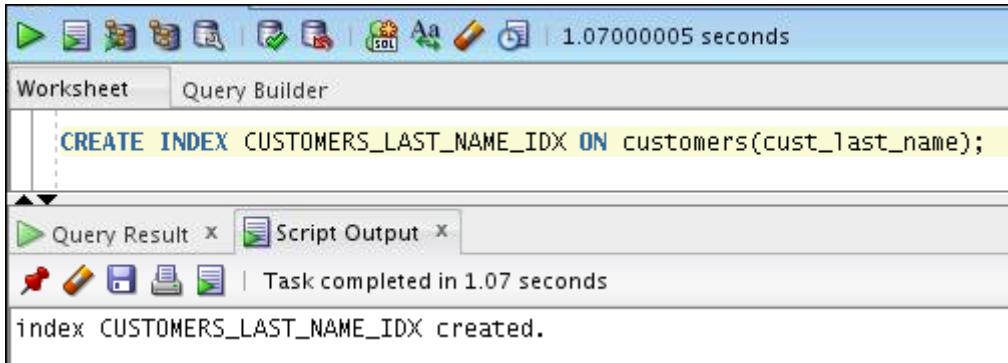
2. Drop all the CUSTOMERS indexes, except its primary key index. Open a terminal window. Connect to sh and drop all the indexes that are currently created on the CUSTOMERS table, except its primary key index, by executing drop_customers_indexes.sqlscript. And create an index on the customers.cust_last_name column. Use the opt_2.sql file.

```
$ cd /home/oracle/labs/solutions/Access_Paths/
$ sqlplus sh/sh

...
Connected to:...

SQL> @drop_customers_indexes.sql
```

```
CREATE INDEX CUSTOMERS_LAST_NAME_IDX ON
customers(cust_last_name);
```



3. Query the index clustering factor of the newly created index. The following query shows that the `customers_last_name_idx` index has a high clustering factor because the clustering factor is significantly more than the number of blocks in the table. Use the `opt_3.sql` file.

```
SELECT index_name, blevel, leaf_blocks, clustering_factor
FROM user_indexes
WHERE table_name= 'CUSTOMERS'
AND index_name= 'CUSTOMERS_LAST_NAME_IDX';
```



4. Create a new copy of the `customers` table, with rows ordered by `cust_last_name`. Use the `opt_4.sql` file.

```
DROP TABLE customers3 PURGE;
CREATE TABLE customers3 AS
  SELECT *
    FROM   customers
   ORDER BY cust_last_name;

DESCRIBE customers3;
```

```
DROP TABLE customers3 PURGE;
CREATE TABLE customers3 AS
  SELECT *
    FROM   customers
   ORDER BY cust_last_name;

DESCRIBE customers3

CREATE TABLE customers3 AS
  SELECT *
    FROM   customers
   ORDER BY cust_last_name
Error report:
SQL Command: table CUSTOMERS3
Failed: Warning: execution completed with warning
describe customers3
Name          Null      Type
-----
CUST_ID        NOT NULL NUMBER
CUST_FIRST_NAME NOT NULL VARCHAR2(20)
CUST_LAST_NAME  NOT NULL VARCHAR2(40)
```

5. Gather statistics on the CUSTOMERS3 table. Use the opt_5.sql file.

```
EXEC DBMS_STATS.GATHER_TABLE_STATS(null,'CUSTOMERS3');
```

```
EXEC DBMS_STATS.GATHER_TABLE_STATS(null,'CUSTOMERS3');

Query Result | Script Output
Task completed in 2.07 seconds
anonymous block completed
```

6. Query the number of rows and blocks in the CUSTOMERS3 table by using the following code. Use the opt_6.sql file.

```
SELECT      TABLE_NAME,  NUM_ROWS,  BLOCKS
FROM        USER_TABLES
WHERE       TABLE_NAME='CUSTOMERS3';
```

The screenshot shows the Oracle SQL Developer interface. The top menu bar includes 'Start Page', 'sh_connection', 'opt_10.sql', and 'opt_4.sql'. The main window has tabs for 'Worksheet' and 'Query Builder', with 'Worksheet' selected. The query editor contains the following SQL code:

```
SELECT      TABLE_NAME,  NUM_ROWS,  BLOCKS
FROM        USER_TABLES
WHERE       TABLE_NAME='CUSTOMERS3';
```

Below the query editor is a 'Query Result' tab. It shows the results of the executed query:

| TABLE_NAME | NUM_ROWS | BLOCKS |
|------------|----------|--------|
| CUSTOMERS3 | 55500 | 1485 |

7. Create an index on the cust_last_name column of the CUSTOMERS3 table. Use the opt_7.sql file.

```
CREATE INDEX CUSTOMERS3_LAST_NAME_IDX ON
customers3(cust_last_name);
```

The screenshot shows the Oracle SQL Developer interface. The top menu bar includes 'sh_connection'. The main window has tabs for 'Worksheet' and 'Query Builder', with 'Worksheet' selected. The query editor contains the following SQL code:

```
CREATE INDEX CUSTOMERS3_LAST_NAME_IDX ON customers3(cust_last_name);
```

Below the query editor is a 'Query Result' tab. It shows the results of the executed query:

```
index CUSTOMERS3_LAST_NAME_IDX created.
```

8. Query the index clustering factor of the customers3_last_name_idx index. The following query shows that the customers3_last_name_idx index has a lower clustering factor. Use the opt_8.sql file.

```
SELECT INDEX_NAME, BLEVEL, LEAF_BLOCKS, CLUSTERING_FACTOR  
FROM   USER_INDEXES  
WHERE  TABLE_NAME = 'CUSTOMERS3'  
AND    INDEX_NAME = 'CUSTOMERS3_LAST_NAME_IDX';
```

The screenshot shows the Oracle SQL Worksheet interface. The query window contains the following SQL statement:

```
SELECT INDEX_NAME, BLEVEL, LEAF_BLOCKS, CLUSTERING_FACTOR  
FROM   USER_INDEXES  
WHERE  TABLE_NAME = 'CUSTOMERS3'  
AND    INDEX_NAME = 'CUSTOMERS3_LAST_NAME_IDX';
```

The results window displays the following data:

| INDEX_NAME | LEVEL | LEAF_BLOCKS | CLUSTERING_FACTOR |
|--------------------------|-------|-------------|-------------------|
| CUSTOMERS3_LAST_NAME_IDX | 1 | 141 | 1453 |

A note at the bottom of the results window states: "All Rows Fetched: 1 in 0.051 seconds".

Note: The table CUSTOMERS3 has the same data as the original customers table, but the index on CUSTOMERS3 has a much lower clustering factor because the data in the table is ordered by cust_last_name. The clustering factor is now about 10 times the number of blocks instead of 70 times.

9. Query the CUSTOMERS table. Use the opt_9.sql file.

```
SELECT cust_first_name, cust_last_name  
FROM   customers  
WHERE  cust_last_name BETWEEN 'Puleo' AND 'Quinn';
```

The screenshot shows the Oracle SQL Worksheet interface. In the 'Worksheet' tab, a query is entered:

```
SELECT cust_first_name, cust_last_name
FROM customers
WHERE cust_last_name BETWEEN 'Puleo' AND 'Quinn';
```

The 'Query Result' tab displays the output:

| CUST_FIRST_NAME | CUST_LAST_NAME |
|-----------------|----------------|
| 1 Vida | Puleo |
| 2 Caresse | Puleo |
| 3 Harriett | Quinlan |
| 4 Madeleine | Quinn |

Execution details: All Rows Fetched: 4 in 0.003 seconds.

10. Execute the following query to display the cursor for the query. Use the opt_10.sql file.

```
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY_CURSOR());
```

The screenshot shows the Oracle SQL Worksheet interface. In the 'Worksheet' tab, the query from the previous step is run:

```
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY_CURSOR());
```

The 'Query Result' tab displays the cursor information:

```

PLAN_TABLE_OUTPUT
1 SQL_ID Ovpc3qhg8xmyn, child number 0
2 -----
3 SELECT cust_first_name, cust_last_name FROM customers WHERE
4 cust_last_name BETWEEN 'Puleo' AND 'Quinn'
5
6 Plan hash value: 2008213504
7
8 -----
9 | Id | Operation          | Name      | Rows   | Bytes | Cost (%CPU) | Time      |
10 -----
11 | 0 | SELECT STATEMENT   |           |        |       | 405 (100)  |           |
12 |/* 1 |  TABLE ACCESS FULL| CUSTOMERS | 2335  | 35025 | 405 (1)    | 00:00:01  |
13 -----
14
15 Predicate Information (identified by operation id):
16 -----
17
18   1 - filter(("CUST_LAST_NAME">>='Puleo' AND "CUST_LAST_NAME"><='Quinn'))
19
```

11. Query the CUSTOMERS3 table. Use the opt_11.sql file.

```
SELECT cust_first_name, cust_last_name
FROM   customers3
WHERE  cust_last_name BETWEEN 'Puleo' AND 'Quinn';
```

| | CUST_FIRST_NAME | CUST_LAST_NAME |
|---|-----------------|----------------|
| 1 | Vida | Puleo |
| 2 | Caresse | Puleo |
| 3 | Harriett | Quinlan |
| 4 | Madeleine | Quinn |

12. Execute the following query to display the cursor for the query. Use the opt_12.sql file.

```
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY_CURSOR());
```

| PLAN_TABLE_OUTPUT |
|--|
| 1 SQL_ID 7qrf03wyn0k0m, child number 0 |
| 2 ----- |
| 3 SELECT cust_first_name, cust_last_name FROM customers3 WHERE |
| 4 cust_last_name BETWEEN 'Puleo' AND 'Quinn' |
| 5 |
| 6 Plan hash value: 843745509 |
| 7 |
| 8 ----- |
| 9 Id Operation Name Rows Bytes Cost (%CPU) Time |
| 10 ----- |
| 11 0 SELECT STATEMENT 71 (100) |
| 12 1 TABLE ACCESS BY INDEX ROWID BATCHED CUSTOMERS3 2335 35025 71 (0) 00:00:01 |
| 13 * 2 INDEX RANGE SCAN CUSTOMERS3_LAST_NAME_IDX 2335 7 (0) 00:00:01 |
| 14 ----- |

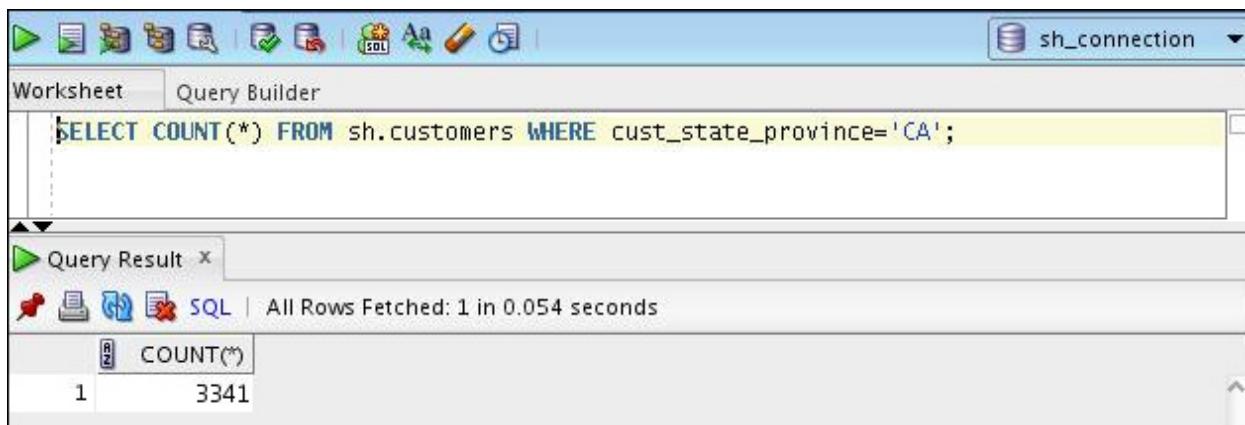
Practice 11-2: Creating Expression Statistics

In this practice, you investigate the use of expression statistics.

1. You can find all the necessary scripts for this practice in your \$HOME/labs/solutions/Optimizer_Statistics directory.

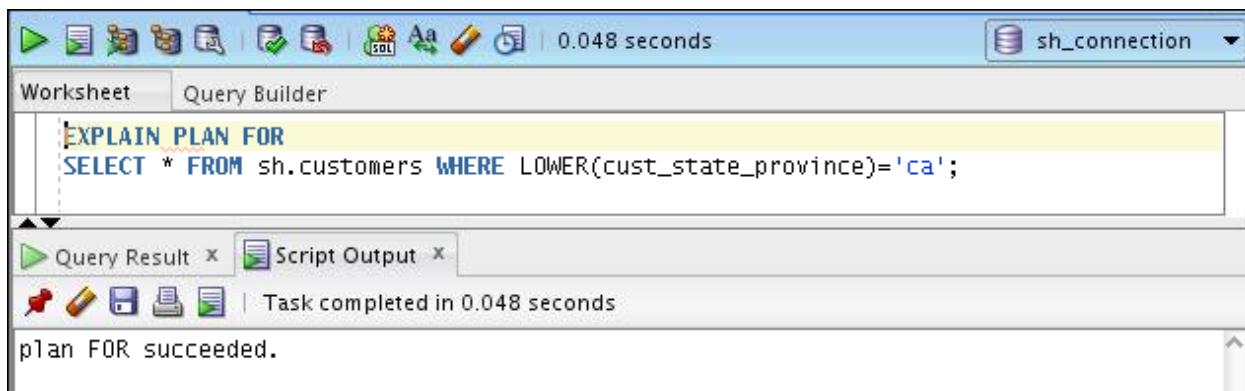
Run the following query on the CUSTOMERS table or use the opt_13.sql file.

```
SELECT COUNT(*) FROM sh.customers WHERE  
cust_state_province='CA';
```



2. Run the EXPLAIN PLAN command for the same query with the LOWER() function applied. Use the opt_14.sql file.

```
EXPLAIN PLAN FOR  
SELECT * FROM sh.customers WHERE  
LOWER(cust_state_province)='ca';
```



3. Run the query to display the plan. Use the opt_15.sql file.

```
select * from table(dbms_xplan.display);
```

The screenshot shows the Oracle SQL Developer interface. In the top navigation bar, there is a connection icon labeled "sh_connection". Below the toolbar, the tabs "Worksheet" and "Query Builder" are visible. A code editor window contains the following SQL command:

```
select * from table(dbms_xplan.display);
```

Below the code editor, there are two tabs: "Explain Plan" and "Query Result". The "Explain Plan" tab is selected, showing the following output:

PLAN_TABLE_OUTPUT

| 1 | Plan hash value: 2008213504 | | | | | | | |
|---|-----------------------------|------------------|-------------------|-----------|-------|-------------|----------|----------|
| 2 | | | | | | | | |
| 3 | | | | | | | | |
| 4 | Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time | |
| 5 | | | | | | | | |
| 6 | 0 | SELECT STATEMENT | | 555 | 112Ki | 423 (1) | 00:00:01 | |
| 7 | * | 1 | TABLE ACCESS FULL | CUSTOMERS | 555 | 112Ki | 423 (1) | 00:00:01 |
| 8 | | | | | | | | |

Note: Because no expression statistics exist for LOWER(cust_state_province)='ca', the optimizer estimate is significantly off. You can use DBMS_STATS procedures to correct these estimates.

4. You can use DBMS_STATS to create statistics for a user-specified expression. Gather table statistics using the following code. Use the opt_16.sql file.

```
BEGIN
  DBMS_STATS.GATHER_TABLE_STATS( 'sh', 'customers', method_opt =>
'FOR ALL COLUMNS SIZE SKEWONLY FOR
COLUMNS(LOWER(cust_state_province)) SIZE SKEWONLY');
END;
/
```

The screenshot shows the Oracle SQL Developer interface. In the top navigation bar, there is a connection icon labeled "sh_connection". Below the toolbar, the tabs "Worksheet" and "Query Builder" are visible. An anonymous block is being run in the worksheet:

```
BEGIN
  DBMS_STATS.GATHER_TABLE_STATS( 'sh', 'customers', method_opt =>
'FOR ALL COLUMNS SIZE SKEWONLY FOR
COLUMNS(LOWER(cust_state_province)) SIZE SKEWONLY');
END;
```

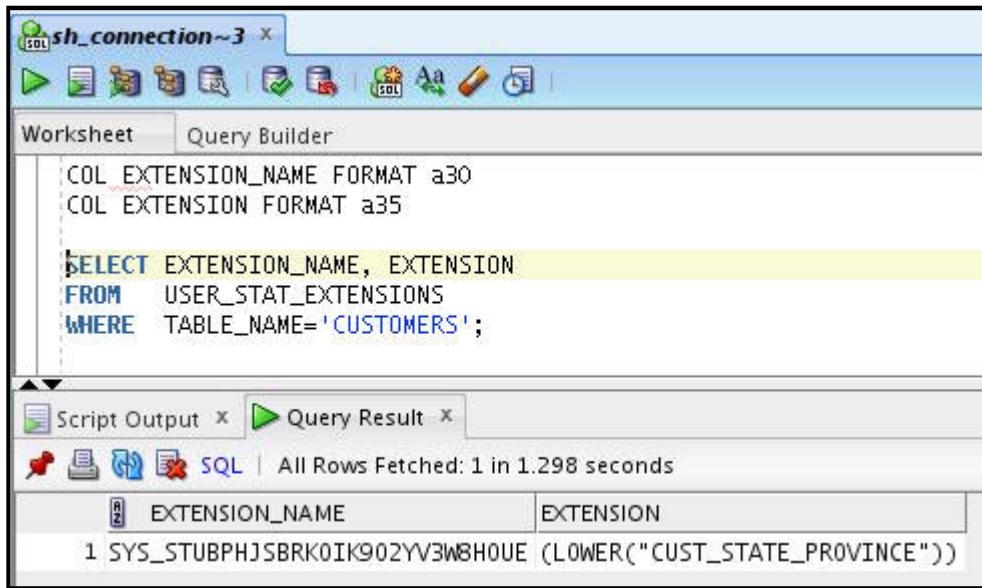
The "Script Output" tab shows the results of the execution:

Task completed in 7.771 seconds
anonymous block completed

5. You can use the database view DBA_STAT_EXTENSIONS and the DBMS_STATS.SHOW_EXTENDED_STATS_NAME function to obtain information about expression statistics. Query the name and definition of the statistics extension by using the following code or use the opt_17.sql file:

```
COL EXTENSION_NAME FORMAT a30
COL EXTENSION FORMAT a35

SELECT EXTENSION_NAME, EXTENSION
FROM   USER_STAT_EXTENSIONS
WHERE  TABLE_NAME='CUSTOMERS';
```



- Run the following code to query the number of distinct values and find whether a histogram has been created for the expression. Use the opt_18.sql file.

```
SELECT e.EXTENSION expression, t.NUM_DISTINCT, t.HISTOGRAM
FROM   USER_STAT_EXTENSIONS e, USER_TAB_COL_STATISTICS t
WHERE  e.EXTENSION_NAME=t.COLUMN_NAME
AND    e.TABLE_NAME=t.TABLE_NAME
AND    t.TABLE_NAME='CUSTOMERS';
```

The screenshot shows the Oracle SQL Developer interface. In the top-left corner, it says "sh_connection~3". Below the title bar are various icons for navigating between tabs and windows. The main area has two tabs: "Worksheet" and "Query Builder". The "Worksheet" tab is active and contains the following SQL code:

```
SELECT e.EXTENSION expression, t.NUM_DISTINCT, t.HISTOGRAM
FROM USER_STAT_EXTENSIONS e, USER_TAB_COL_STATISTICS t
WHERE e.EXTENSION_NAME=t.COLUMN_NAME
AND e.TABLE_NAME=t.TABLE_NAME
AND t.TABLE_NAME='CUSTOMERS';
```

Below the code, there are two tabs: "Script Output" and "Query R...". The "Script Output" tab is active and displays the results of the query:

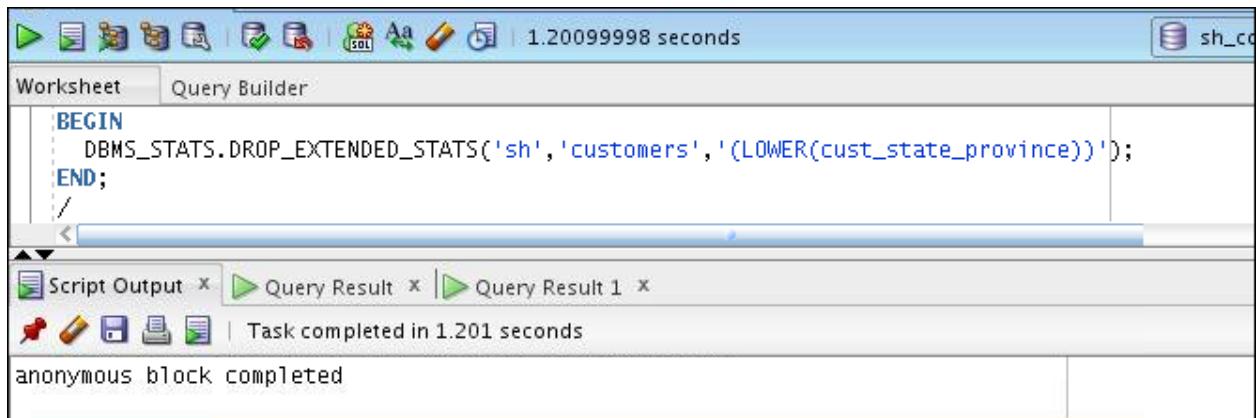
| EXPRESSION | NUM_DISTINCT | HISTOGRAM |
|----------------------------------|--------------|-----------|
| 1 (LOWER("CUST_STATE_PROVINCE")) | 145 | FREQUENCY |

At the bottom of the "Script Output" tab, it says "All Rows Fetched: 1 in 1.569 seconds".

7. You created extended statistics for the LOWER(cust_state_province) expression. Now to drop the expression statistics, use the following code. Use the opt_19.sql file.

```
BEGIN

DBMS_STATS.DROP_EXTENDED_STATS('sh', 'customers', '(LOWER(cust_state_province))');
END;
/
```



8. Drop all the CUSTOMERS indexes, except its primary key index. Open a terminal window. Connect to sh and drop all the indexes that are currently created on the CUSTOMERS table, except its primary key index, by executing `drop_customers_indexes.sql`. And create an index on the `customers.cust_last_name` column. Use the `opt_2.sql` file.

```
$ cd /home/oracle/labs/solutions/Access_Paths/
$ sqlplus sh/sh

...
Connected to:...

SQL> @drop_customers_indexes.sql
```

Practice 11-3: Enabling Automatic Statistics Gathering (Optional)

In this practice, you manipulate object statistics to see the effect of your SQL statements on the execution plans.

Note: All the scripts needed for this practice can be found in your \$HOME/labs/solutions/Automatic_Gather_Stats directory.

1. The environment for this practice has been set up with the ags_setup.sh script. This script created a user called AGS that you use throughout this practice. The script also created a table called EMP and an index. The script is listed as follows:

```
-----  
#!/bin/bash  
  
cd /home/oracle/labs/solutions/Automatic_Gather_Stats  
  
sqlplus / as sysdba @ags_setup.sql  
  
-----  
  
set echo on  
  
drop user ags cascade;  
  
create user ags identified by ags;  
  
grant dba to ags;  
  
connect ags/ags  
  
drop table emp purge;  
  
create table emp  
(  
empno number,  
ename varchar2(20),  
phone varchar2(20),  
deptno number  
);  
  
insert into emp  
with tdata as  
(select rownum empno  
from all_objects  
where rownum <= 1000)  
select rownum,
```

```
dbms_random.string ('u', 20),
dbms_random.string ('u', 20),
case
    when rownum/100000 <= 0.001 then mod(rownum, 10)
    else 10
end
from tdata a, tdata b
where rownum <= 100000;

commit;

create index emp_i1 on emp(deptno);

exit;
```

2. Execute the ags_start.sql script in SQL*Plus from the \$HOME/labs/solutions/Automatic_Gather_Stats directory.

```
$ cd $HOME/labs/solutions/Automatic_Gather_Stats
$ ./ags_start.sh

...
SQL>
SQL> connect / as sysdba
Connected.
SQL>
SQL> alter system flush shared_pool;

System altered.

SQL>
SQL> --
SQL> -- Turn off AUTOTASK
SQL> --
SQL>
SQL> alter system set "_enable_automatic_maintenance"=0;

System altered.

SQL>
SQL> exec dbms_stats.delete_schema_stats('AGS');

PL/SQL procedure successfully completed.
```

```
SQL>
SQL> exit;
Disconnected ...
$
```

3. Connect as the ags user in a SQL*Plus session.

```
$ sqlplus ags/ags
...
SQL>
```

4. Create a new connection in SQL Developer for the ags user with the following properties:

Name: ags_connection

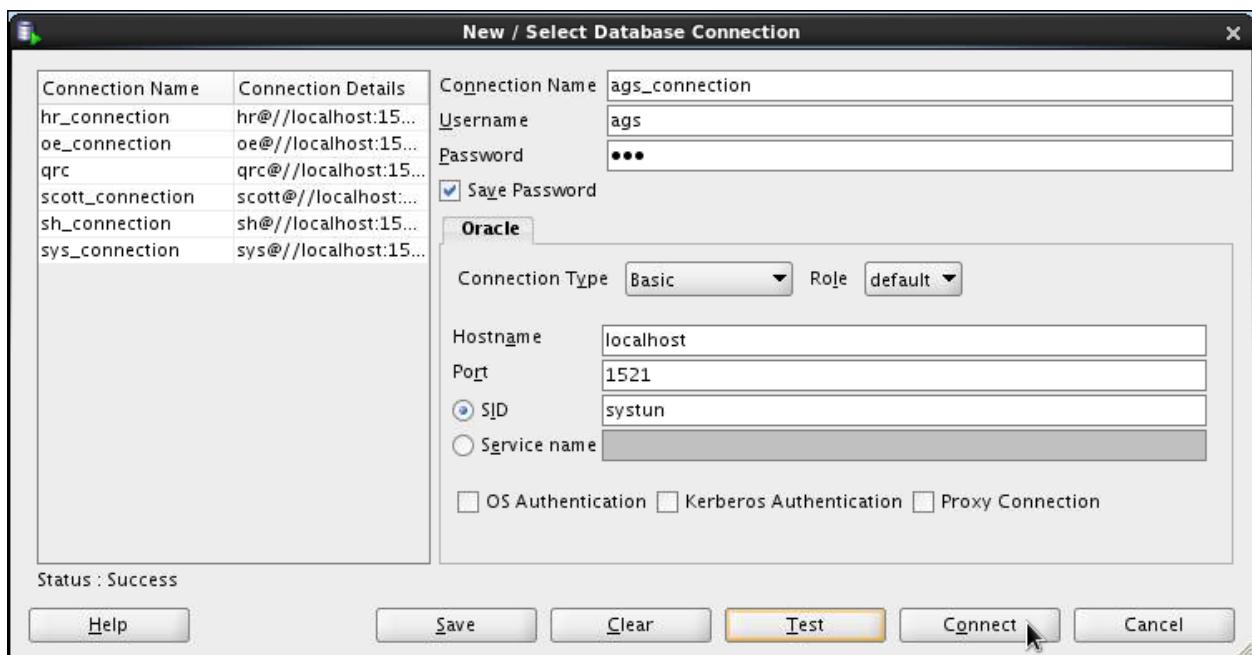
User: ags

Password: ags

SID: systun

Click Test.

Click Connect.



5. Using ags_connection, determine the distribution of the deptno values found in the EMP table. What do you observe?

- a. Open and execute the show_deptno_distribution.sql file in the \$HOME/labs/solutions/Automatic_Gather_Stats directory. You can see that

there are 11 department values, each repeating 0.01% of the time, except value 10 that repeats 99.9% of the time.

The screenshot shows the Oracle SQL Developer interface. The top window is titled 'show_deptno_distribution.sql' and contains the following SQL code:

```
select deptno, count(*) cnt_per_deptno, (count(*)*100)/nr deptno_percent
from emp, (select max(empno) nr
            from emp)
group by deptno, nr
order by deptno;
```

The bottom window is titled 'Script Output' and displays the results of the query:

| DEPTNO | CNT_PER_DEPTNO | DEPTNO_PERCENT |
|--------|----------------|----------------|
| 0 | 10 | 0.01 |
| 1 | 10 | 0.01 |
| 2 | 10 | 0.01 |
| 3 | 10 | 0.01 |
| 4 | 10 | 0.01 |
| 5 | 10 | 0.01 |
| 6 | 10 | 0.01 |
| 7 | 10 | 0.01 |
| 8 | 10 | 0.01 |
| 9 | 10 | 0.01 |
| 10 | 99900 | 99.9 |

Below the table, it says '11 rows selected'.

6. Determine if there are histograms available on any column of the EMP table. What do you observe?
 - a. Open the check_emp_histogram.sql file and execute it in SQL Developer. Currently, there are no histograms defined on any column of the EMP table.

The screenshot shows the Oracle SQL Developer interface. In the top navigation bar, there are tabs for 'ags_connection', 'show_deptno_distribution.sql', and 'check_emp_histogram.sql'. The 'check_emp_histogram.sql' tab is active. Below the tabs is a toolbar with various icons. The main area has two tabs: 'Worksheet' and 'Query Builder', with 'Worksheet' selected. A SQL query is entered in the worksheet:

```
select column_name, histogram, num_buckets
from user_tab_columns
where table_name='EMP';
```

Below the worksheet is a 'Script Output' window showing the results of the query:

| COLUMN_NAME | HISTOGRAM | NUM_BUCKETS |
|-------------|-----------|-------------|
| EMPNO | NONE | |
| ENAME | NONE | |
| PHONE | NONE | |
| DEPTNO | NONE | |

A message at the bottom of the output window says 'Task completed in 0.057 seconds'.

7. Determine if you currently have statistics gathered on your EMP table. What do you observe?
 - a. In the SQL Developer navigator pane, click the Connections tab. Expand ags_connection, and then the Tables node and the EMP table node. Select the EMP table node.
 - b. On the EMP tab, click the Statistics subtab.
 - c. Currently, there are no statistics gathered on your EMP table.

The screenshot shows the Oracle SQL Developer interface with the 'EMP' table selected. The top navigation bar has tabs for 'ags_connection', 'show_deptno_distribution.sql', 'check_emp_histogram.sql', and 'EMP'. The 'EMP' tab is active. Below the tabs is a toolbar with icons for Actions, Columns, Data, Constraints, Grants, Statistics, Triggers, Flashback, Dependencies, Details, Partitions, Indexes, and SQL. The 'Statistics' tab is selected. The main area displays a table with columns 'Name' and 'Value'.

| Name | Value |
|---------------------|--------|
| NUM_ROWS | (null) |
| BLOCKS | (null) |
| AVG_ROW_LEN | (null) |
| SAMPLE_SIZE | (null) |
| LAST_ANALYZED | (null) |
| LAST_ANALYZED_SINCE | (null) |

8. Determine if you currently have statistics gathered on the index that is created on top of the EMP table. What do you observe?
 - a. Click the Indexes subtab of the EMP pane.

The screenshot shows the Oracle SQL Developer interface with the 'EMP' table selected. The top navigation bar has tabs for 'ags_connection', 'show_deptno_distribution.sql', 'check_emp_histogram.sql', and 'EMP'. The 'EMP' tab is active. Below the tabs is a toolbar with icons for Actions, Columns, Data, Constraints, Grants, Statistics, Triggers, Flashback, Dependencies, Details, Partitions, Indexes, and SQL. The 'Indexes' tab is selected. The main area displays a table with columns: INDEX_OWNER, INDEX_NAME, UNIQUENESS, STATUS, INDEX_TYPE, TEMPORARY, PARTITIONED, FUNCIDX_STATUS, JOIN_INDEX, and C.

| INDEX_OWNER | INDEX_NAME | UNIQUENESS | STATUS | INDEX_TYPE | TEMPORARY | PARTITIONED | FUNCIDX_STATUS | JOIN_INDEX | C |
|-------------|------------|------------|--------|------------|-----------|-------------|----------------|------------|--------|
| AGS | EMP_I1 | NONUNIQUE | VALID | NORMAL | N | NO | (null) | NO | DEPTNO |

- b. In the Connections navigator pane, under ags_connection, expand the indexes node, and then select the EMP_I1 index. In the EMP_I1 pane, click the Statistics subtab. Currently, EMP_I1 has no statistics gathered.

The screenshot shows the 'Statistics' tab of the Oracle SQL Developer interface for the 'ags_connection' session. The tab bar includes 'Columns', 'Details', 'Statistics', 'Partitions', and 'SQL'. The title bar shows three open tabs: 'show_deptno_distribution.sql', 'check_emp_histogram.sql', and 'EMP_I1'. The 'EMP_I1' tab is active, displaying a table of optimizer statistics. The table has two columns: 'Name' and 'Value'. The 'Name' column lists various statistics starting from 1 and ending at 24. The 'Value' column contains mostly '(null)' entries, except for 'OWNER' which is 'AGS', 'INDEX_NAME' which is 'EMP_I1', and 'OBJECT_TYPE' which is 'INDEX'. The row for 'SAMPLE_SIZE' is highlighted with a red box.

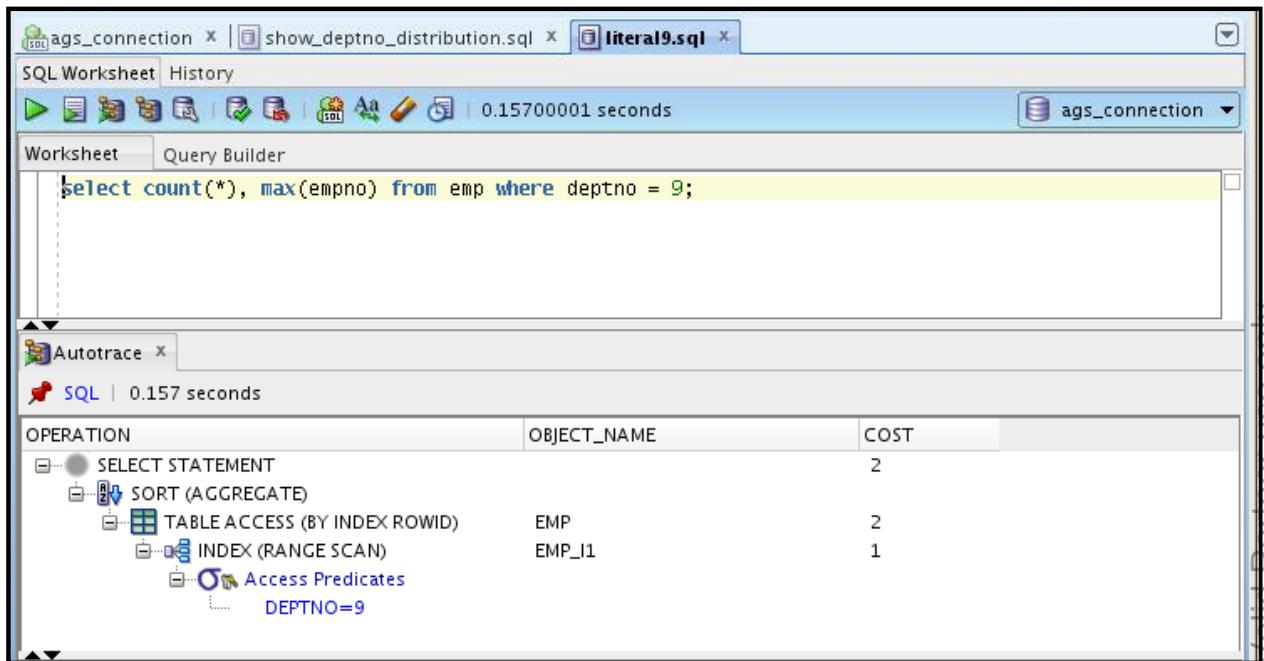
| Name | Value |
|----------------------------|--------|
| 1 OWNER | AGS |
| 2 INDEX_NAME | EMP_I1 |
| 3 TABLE_OWNER | AGS |
| 4 TABLE_NAME | EMP |
| 5 PARTITION_NAME | (null) |
| 6 PARTITION_POSITION | (null) |
| 7 SUBPARTITION_NAME | (null) |
| 8 SUBPARTITION_POSITION | (null) |
| 9 OBJECT_TYPE | INDEX |
| 10 BLEVEL | (null) |
| 11 LEAF_BLOCKS | (null) |
| 12 DISTINCT_KEYS | (null) |
| 13 AVG_LEAF_BLOCKS_PER_KEY | (null) |
| 14 AVG_DATA_BLOCKS_PER_KEY | (null) |
| 15 CLUSTERING_FACTOR | (null) |
| 16 NUM_ROWS | (null) |
| 17 AVG_CACHED_BLOCKS | (null) |
| 18 AVG_CACHE_HIT_RATIO | (null) |
| 19 SAMPLE_SIZE | (null) |
| 20 LAST_ANALYZED | (null) |
| 21 GLOBAL_STATS | NO |
| 22 USER_STATS | NO |
| 23 STATTYPE_LOCKED | (null) |
| 24 STALE_STATS | (null) |

9. Autotrace the following two statements and determine their execution plans:

```
select count(*), max(empno) from emp where deptno = 9;  
select count(*), max(empno) from emp where deptno = 10;
```

These statements are in the literal9.sql and literal10.sql files. What do you observe and why?

- a. Open the literal9.sql file and autotrace in ags_connection. You see that for deptno = 9, the optimizer decided to use the index.



- b. Open the literal10.sql file and autotrace in ags_connection. Notice that for deptno = 10, the optimizer decided to do TABLE ACCESS FULL. The optimizer determined the correct plan in both cases. This is because dynamic statistics was used and there were no statistics gathered on either object.



10. Confirm your assumption from the previous step.
- On the ags_connection tab, enter the following query and execute:
- ```
SELECT * FROM v$parameter WHERE NAME LIKE '%dynamic%';
```
- Note:** In SQL Developer, there is only one session for a connection even though there may be multiple windows using that session.

The screenshot shows the Oracle SQL Developer interface. In the top tab bar, there are several tabs: 'ags\_connection' (selected), 'show\_deptno\_distribution.sql', 'literal9.sql', and 'literal10.sql'. The main area is a 'Worksheet' tab showing the following SQL command:

```
SELECT * FROM v$parameter WHERE NAME LIKE '%dynamic%';
```

Below the worksheet, the 'Script Output' tab displays the results of the query. It shows one row in a table:

| NUM  | NAME                       | TYPE | VALUE |
|------|----------------------------|------|-------|
| 1801 | optimizer_dynamic_sampling | 3    | 2     |

A note below the table says 'Task completed in 0.022 seconds'.

11. Turn off dynamic statistics.

- Set the parameter for your session to 0. Use the following command.

```
ALTER session SET optimizer_dynamic_sampling = 0;
```

The screenshot shows the Oracle SQL Developer interface. The top tab bar has tabs for 'ags\_connection' (selected), 'show\_deptno\_distribution.sql', 'literal9.sql', and 'literal10.sql'. The 'Worksheet' tab contains the following SQL command:

```
ALTER session SET optimizer_dynamic_sampling = 0;
```

The 'Script Output' tab shows the result of the command:

```
session SET altered.
```

A note below the output says 'Task completed in 0.005 seconds'.

- Confirm that the parameter has changed. Open SQL\_HISTORY, select the select \* from V\$parameter command, click Replace, and execute.



The screenshot shows the Oracle SQL Developer interface with three main panes:

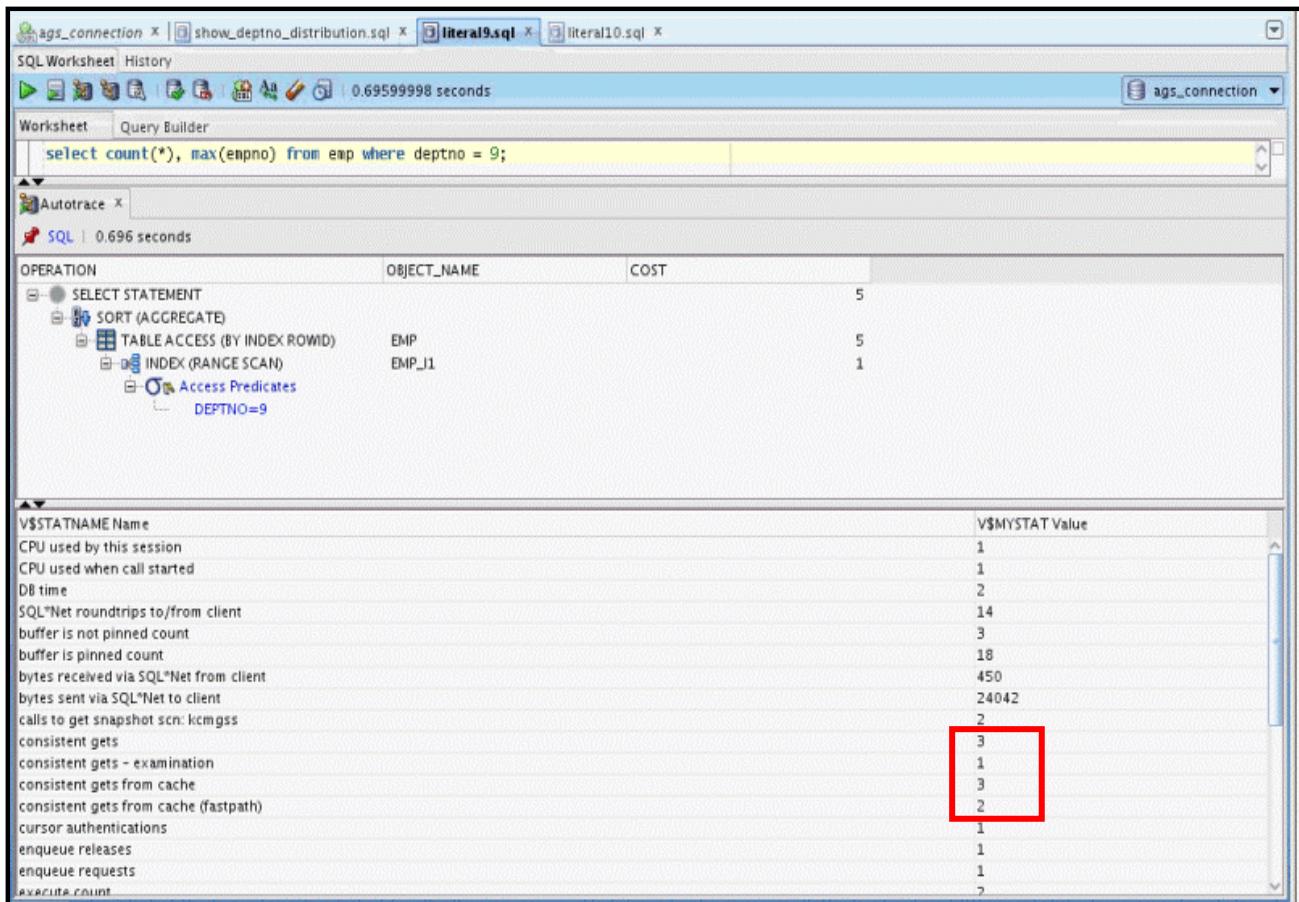
- Worksheet:** Displays the SQL query: `SELECT * FROM v$parameter WHERE NAME LIKE '%dynamic%';`
- Script Output:** Shows the result of the query execution: "Task completed in 0.029 seconds". Below it is a table with one row:

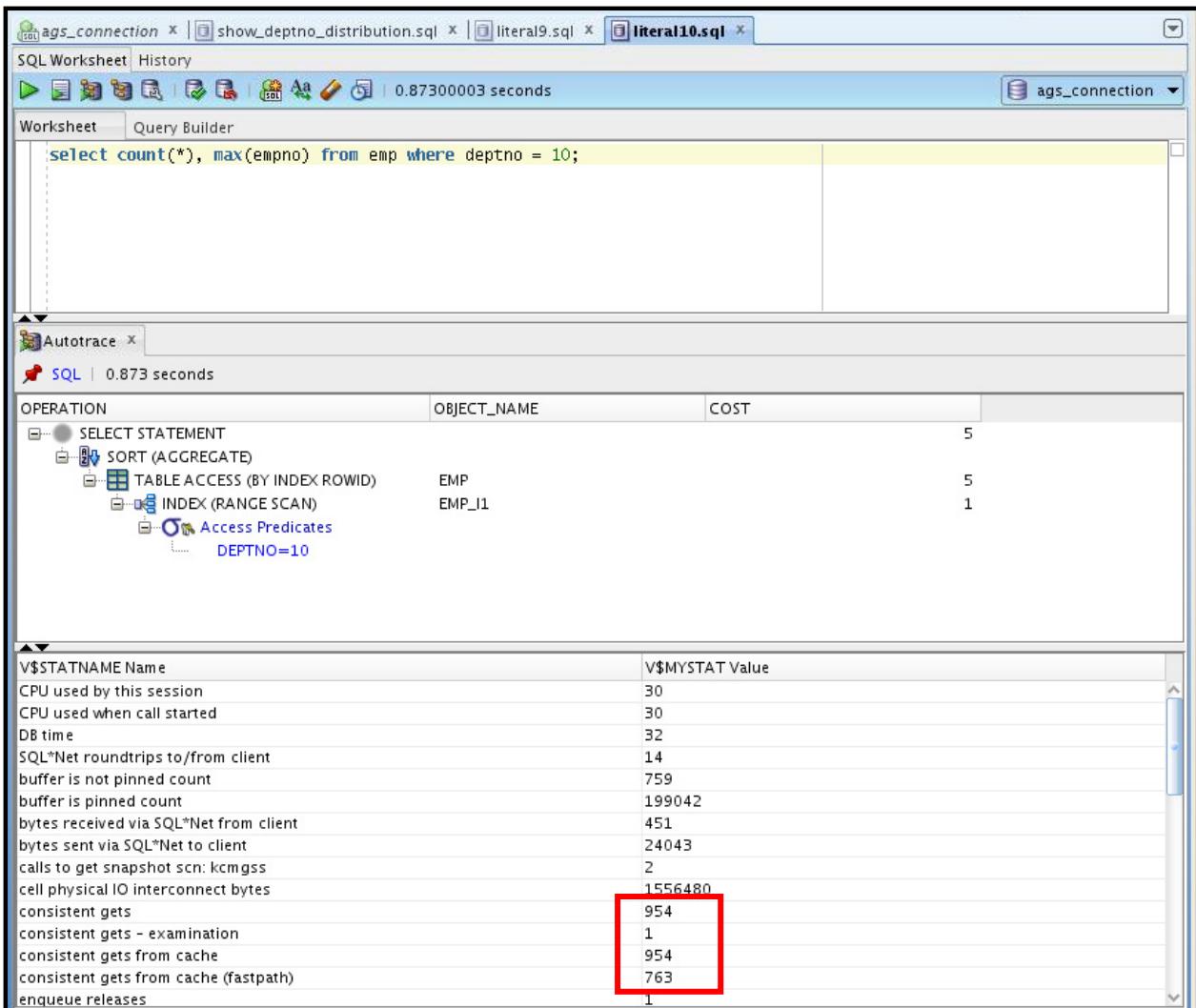
| NUM  | NAME                       | TYPE | VALUE |
|------|----------------------------|------|-------|
| 1801 | optimizer_dynamic_sampling | 3    | 0     |

- SQL History:** A grid showing the history of SQL statements executed. The first statement in the history is the one shown in the Worksheet.

12. Autotrace both literal9.sql and literal10.sql again. What do you observe and why?

Because dynamic statistics is not used, the optimizer cannot use any real-time statistics. It uses default statistics that are not a good choice for the second statement. The cost is estimated to be the same, but when you compare the actual statistics, you can see the difference.





13. Reset dynamic statistics as it was at the beginning of this practice. Use the following command:

|                                                   |
|---------------------------------------------------|
| ALTER session SET optimizer_dynamic_sampling = 2; |
|---------------------------------------------------|

The screenshot shows the Oracle SQL Developer interface. The title bar says 'ags\_connection'. The main area is a 'Worksheet' tab with the following SQL command:

```
ALTER session SET optimizer_dynamic_sampling = 2;
```

Below the worksheet is a 'Script Output' tab showing the result:

```
session SET altered.
```

14. Set the following wrong statistic values for your EMP table:

- Set its number of rows to 10.
- Set its number of blocks to 5.

Open the `set_wrong_stats.sql` file and execute.

The screenshot shows the Oracle SQL Developer interface. The title bar says 'ags\_connection' and has a tab for 'set\_wrong\_stats.sql'. The main area is a 'Worksheet' tab with the following SQL code:

```
exec dbms_stats.set_table_stats('AGS', 'EMP',null,null,null,10,5);
```

Below the worksheet is a 'Script Output' tab showing the result:

```
anonymous block completed
```

15. Verify that you modified the statistics of the EMP table correctly.

- In the navigator pane, select the EMP table under `ags_connection`. On the EMP tab, click the Statistics subtab.

Screenshot of Oracle SQL Developer showing the Statistics tab for the EMP table. The table lists various statistics with their values. The 'NUM\_ROWS' and 'BLOCKS' rows are highlighted with red boxes.

| Name                | Value     |
|---------------------|-----------|
| NUM_ROWS            | 10        |
| BLOCKS              | 5         |
| AVG_ROW_LEN         | 100       |
| SAMPLE_SIZE         | 2000      |
| LAST_ANALYZED       | 19-JUL-12 |
| LAST_ANALYZED_SINCE | 19-JUL-12 |

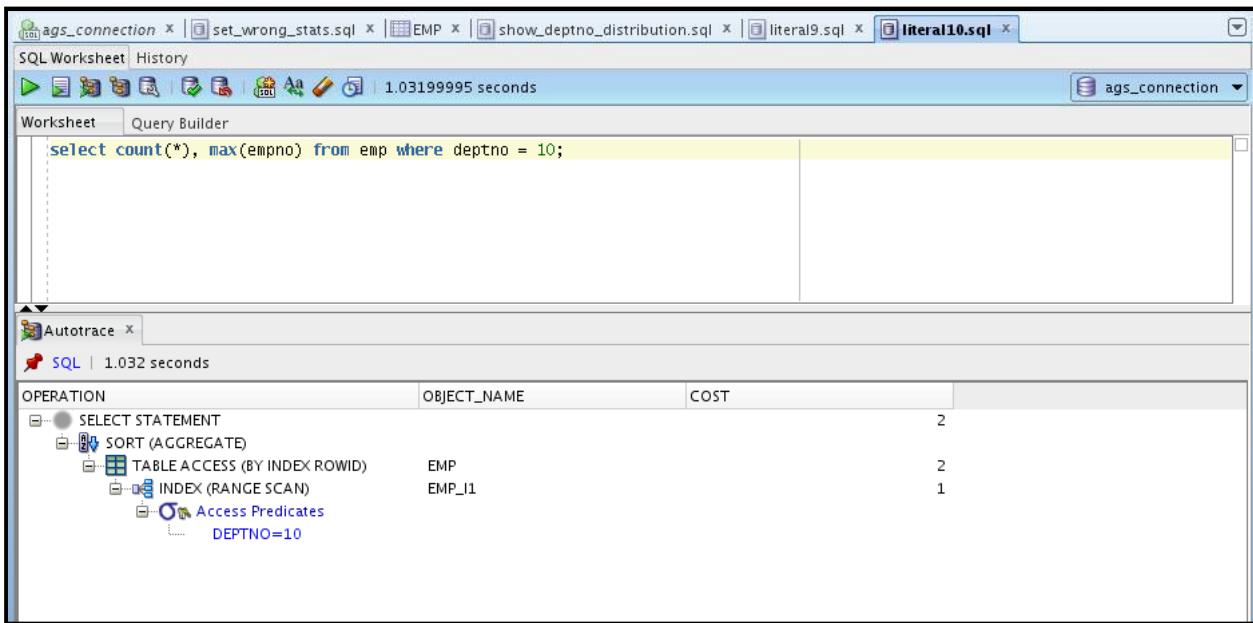
16. Autotrace both literal9.sql and literal10.sql again. What do you observe and why?
- Because there are statistics defined on the EMP table, the optimizer uses them, and not dynamic statistics. However, because the statistics are incorrect, the generated plans are also incorrect, at least for the second statement. This is noticeable in the execution time.

Screenshot of Oracle SQL Developer showing the Autotrace results for the query:

```
select count(*), max(empno) from emp where deptno = 9;
```

The Autotrace output shows the following execution plan:

| OPERATION                     | OBJECT_NAME | COST |
|-------------------------------|-------------|------|
| SELECT STATEMENT              |             | 2    |
| SORT (AGGREGATE)              |             |      |
| TABLE ACCESS (BY INDEX ROWID) | EMP         | 2    |
| INDEX (RANGE SCAN)            | EMP_I1      | 1    |
| Access Predicates             |             |      |
| DEPTNO=9                      |             |      |



17. Make sure that you manually gather statistics on your EMP table and its corresponding index. Enter the following or execute the `gather_stats_manually.sql` script:

```
exec dbms_stats.gather_table_stats('AGS', 'EMP', cascade => true);
```



18. Make sure that statistics were gathered on your objects. Select the EMP table in the navigator pane. Click the Statistics subtab on the EMP tab. Click the Refresh button to ensure that the latest statistics are displayed. Notice that Column Statistics are populated.

The screenshot shows the Oracle SQL Developer interface. The top menu bar has tabs for 'Columns', 'Data', 'Constraints', 'Grants', 'Statistics', 'Triggers', 'Flashback', 'Dependencies', 'Details', 'Partitions', 'Indexes', and 'SQL'. The 'Statistics' tab is selected. Below the menu is a toolbar with icons for refresh, search, and actions. A table titled 'Column Statistics' is displayed, showing the following data:

| OWNER | TABLE_NAME | COLUMN_NAME | NUM_DISTINCT | LOW_VALUE                                | HIGH_VALUE                           |
|-------|------------|-------------|--------------|------------------------------------------|--------------------------------------|
| AGS   | EMP        | EMPNO       | 100000       | C102                                     | C30B                                 |
| AGS   | EMP        | ENAME       | 100000       | 414141424D4C424449525450594D564F48545346 | 5A5A5A5958524A594A4F4E555852425A4149 |
| AGS   | EMP        | PHONE       | 100000       | 41414142484541584D4D46564252564A5258494E | 5A5A5A5A4851485344554E564A4F444E454A |
| AGS   | EMP        | DEPTNO      | 1180         |                                          | C10B                                 |

19. Autotrace both literal9.sql and literal10.sql again. What do you observe and why?

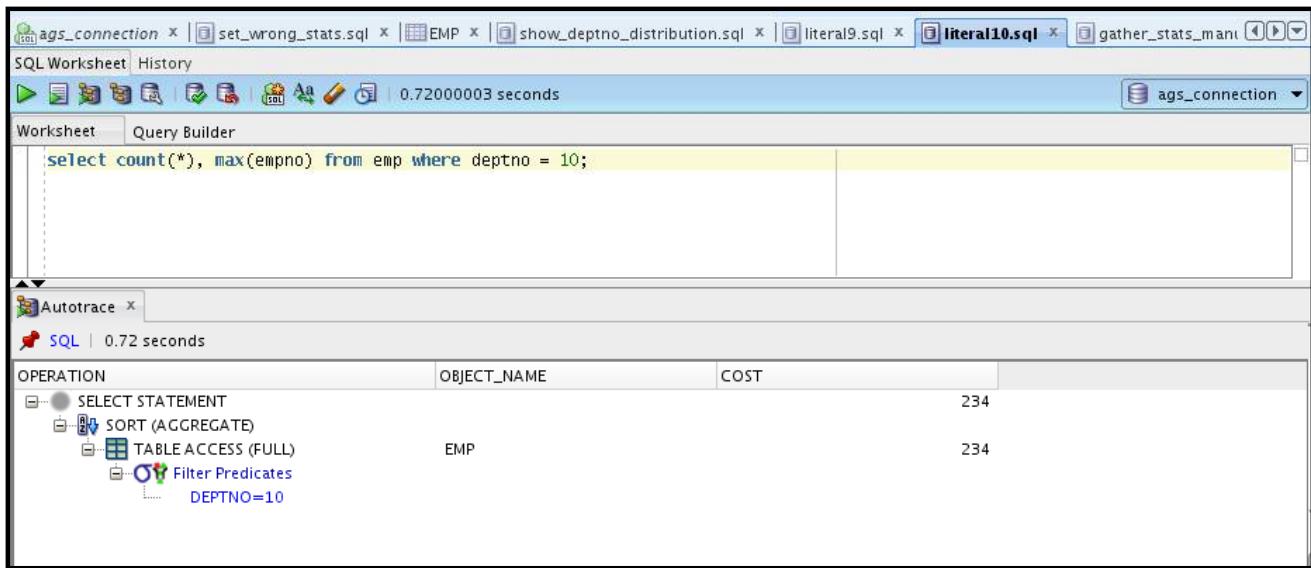
- Because statistics were correctly gathered on both objects, the optimizer is able to use the correct execution plans for both statements.

The screenshot shows the Oracle SQL Developer interface with the 'literal9.sql' tab selected. The top bar shows the connection name 'ags\_connection'. Below the bar is a toolbar with various icons. The main area shows the SQL query:

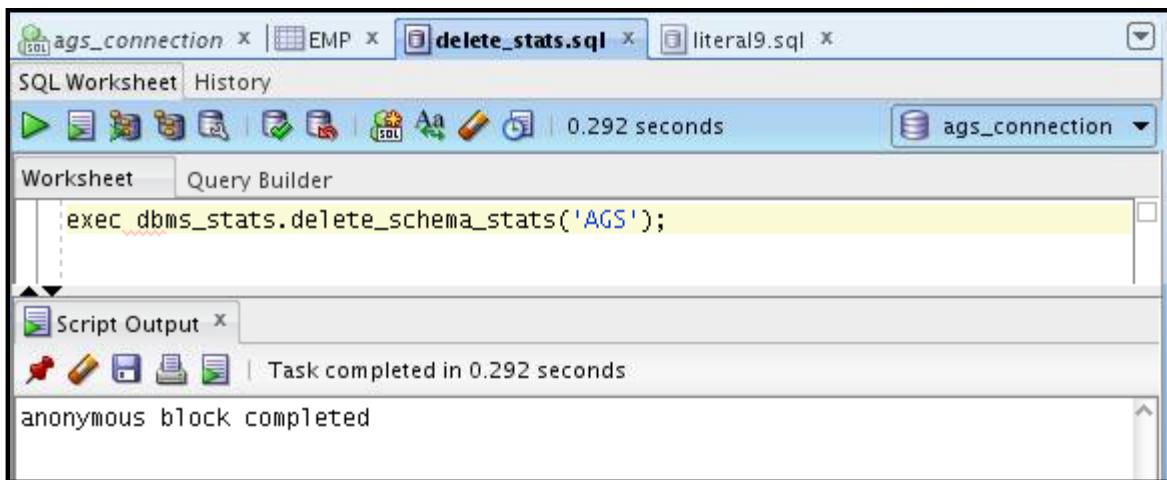
```
select count(*), max(empno) from emp where deptno = 9;
```

Below the query is the 'Autotrace' window, which displays the execution plan:

| OPERATION                     | OBJECT_NAME | COST |
|-------------------------------|-------------|------|
| SELECT STATEMENT              |             | 2    |
| SORT (AGGREGATE)              |             |      |
| TABLE ACCESS (BY INDEX ROWID) | EMP         | 2    |
| INDEX (RANGE SCAN)            | EMP_I1      | 1    |
| Access Predicates             |             |      |
| DEPTNO=9                      |             |      |



20. Delete all the statistics that were previously generated on the EMP tab and the EMP\_I1 index. Open and execute the delete\_stats.sql script.



21. Verify that you no longer have statistics gathered on both objects.

- In the navigator pane, select the EMP table and click the Statistics subtab on the EMP tab, and refresh.

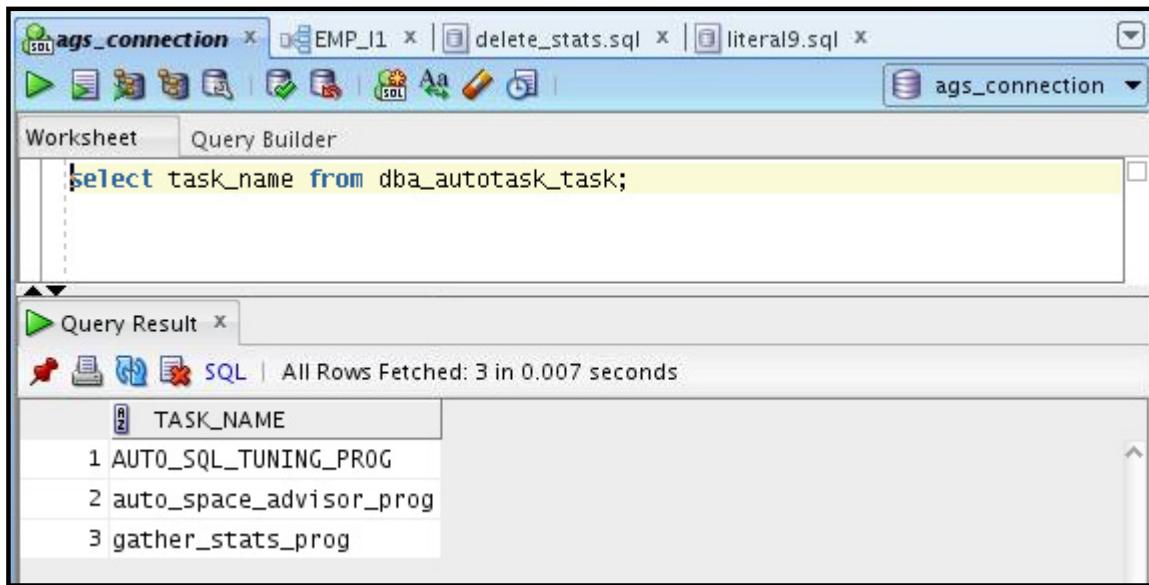
The screenshot shows the Oracle SQL Developer interface. The top menu bar has tabs for 'Columns', 'Data', 'Constraints', 'Grants', 'Statistics' (which is selected), 'Triggers', 'Flashback', 'Dependencies', 'Details', 'Partitions', 'Indexes', and 'SQL'. Below the menu is a toolbar with icons for refresh, edit, and search, followed by a dropdown menu labeled 'Actions...'. The main pane displays a table with columns 'Name' and 'Value'. The rows show statistics for the EMP table: NUM\_ROWS (null), BLOCKS (null), AVG\_ROW\_LEN (null), SAMPLE\_SIZE (null), LAST\_ANALYZED (null), and LAST\_ANALYZED\_SINCE (null). At the bottom of the main pane, there is a 'Column Statistics' section with a 'Refresh' button set to 0. Below this are various statistical parameters: OWNER, TABLE\_NAME, COLUMN\_NAME, NUM\_DISTINCT, LOW\_VALUE, HIGH\_VALUE, DENSITY, NUM\_NULLS, NUM\_BINS, LAST\_ANALYZED, SAMPLE\_SIZE, and GLOBAL\_STATS.

- b. In the navigator pane, select the `EMP_I1` index, click the Statistics subtab on the `EMPI1` tab, and refresh.

| Name                       | Value  |
|----------------------------|--------|
| 1 OWNER                    | AGS    |
| 2 INDEX_NAME               | EMP_I1 |
| 3 TABLE_OWNER              | AGS    |
| 4 TABLE_NAME               | EMP    |
| 5 PARTITION_NAME           | (null) |
| 6 PARTITION_POSITION       | (null) |
| 7 SUBPARTITION_NAME        | (null) |
| 8 SUBPARTITION_POSITION    | (null) |
| 9 OBJECT_TYPE              | INDEX  |
| 10 LEVEL                   | (null) |
| 11 LEAF_BLOCKS             | (null) |
| 12 DISTINCT_KEYS           | (null) |
| 13 AVG_LEAF_BLOCKS_PER_KEY | (null) |
| 14 AVG_DATA_BLOCKS_PER_KEY | (null) |
| 15 CLUSTERING_FACTOR       | (null) |
| 16 NUM_ROWS                | (null) |
| 17 AVG_CACHED_BLOCKS       | (null) |
| 18 AVG_CACHE_HIT_RATIO     | (null) |
| 19 SAMPLE_SIZE             | (null) |
| 20 LAST_ANALYZED           | (null) |
| 21 GLOBAL_STATS            | NO     |
| 22 USER_STATS              | NO     |
| 23 STATTYPE_LOCKED         | (null) |
| 24 STALE_STATS             | (null) |

22. How would you determine the list of automated tasks that exist on your database?
- You can use Enterprise Manager Database Control by navigating to the Automated Maintenance Tasks page (Home > Server > Automated Maintenance Tasks). On the Automated Maintenance Tasks page, you can see the three automated tasks implemented by default on your database.
  - Another possibility is to use the `DBA_AUTOTASK_TASK` table as shown by the following statement:

```
select task_name from dba_autotask_task;
```



23. You now want to observe the effects of the automatic statistics gathering feature of your database. However, you do not want to wait until the database automatically opens the next maintenance window. From SQL Developer, open and execute the `run_ags_pl.sql` script. This script forces the execution of the automatic statistics gathering task.
- If you do not already have a `sys_connection`, create a connection as the `sysdba` user.  
Name: `sys_connection`  
Username: `sys`  
Password: `oracle_4U`  
Role: `sysdba`  
SID: `systun`
  - Test and save the connection.
  - Open the `run_ags_pl.sql` file.

The screenshot shows a SQL Worksheet window titled "run\_ags\_plsql". The connection is set to "sys\_connection". The code in the worksheet is as follows:

```
SET SERVEROUTPUT ON
DECLARE
 WINDOW VARCHAR2 (20);
BEGIN
 DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT;

 SELECT UPPER(TO_CHAR(SYSDATE,'fmday'))||'_WINDOW' INTO WINDOW FROM DUAL;
 DBMS_OUTPUT.PUT_LINE('Window is: '||WINDOW);

 -- Open the corresponding maintenance window, but with other clients disabled
 --
 EXECUTE IMMEDIATE 'alter system set "_enable_automatic_maintenance"=1';

 dbms_auto_task_admin.disable('auto space advisor', null, window);
 dbms_auto_task_admin.disable('sql tuning advisor', null, window);

 dbms_scheduler.open_window(window, null, true);
 --
 -- Close the maintenance window when auto optimizer stats collection is done
 --
 DBMS_LOCK.SLEEP(120);

 dbms_scheduler.close_window(window);

 EXECUTE IMMEDIATE 'alter system set "_enable_automatic_maintenance"=0';
 --
 -- Re-enable the other guys so they look like they are enabled in EM.
 -- Still they will be disabled because we have set the underscore.
 --
 dbms_auto_task_admin.enable('auto space advisor', null, window);
 dbms_auto_task_admin.enable('sql tuning advisor', null, window);

end;
```

- d. Open the Dbms Output pane (View > Dbms Output) and click the green plus sign to connect to sys\_connection.

The screenshot shows the Oracle SQL Developer interface. The top menu bar has tabs for 'ags\_connection', 'run\_ags\_pl.sql', 'sys\_connection', 'EMP\_J1', and 'delete\_stats.sql'. The main window is titled 'Worksheet' and contains the following PL/SQL code:

```
Set SERVEROUTPUT ON
DECLARE
 WINDOW VARCHAR2 (20);
BEGIN
 DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT;
 SELECT UPPER(TO_CHAR(SYSDATE, 'fmday'))||'_WINDOW' INTO WINDOW FROM DUAL;
 DBMS_OUTPUT.PUT_LINE('Window is: '|| WINDOW);

```

Below the worksheet is a 'Dbms Output' pane with a buffer size of 20000. It displays the output:

```
Window is: THURSDAY_WINDOW
```

- e. Execute the `run_ags_pl.sql` script as the `sys` user. Use `sys_connection`. This script takes two minutes to run.

The screenshot shows the Oracle SQL Developer interface. The top menu bar has tabs for 'ags\_connection', 'run\_ags\_pl.sql', 'sys\_connection', 'EMP\_J1', and 'delete\_stats.sql'. The main window is titled 'Worksheet' and contains the same PL/SQL code as the previous screenshot. Below the worksheet is a 'Script Output' pane which shows a message: 'Task completed in 120.997 seconds'. The 'Dbms Output' pane below it also displays the output:

```
anonymous block completed
Window is: THURSDAY_WINDOW
```

- f. Close the Dbms Output pane.

24. Navigate to the `EMP` table and view the statistics. Be sure to refresh the view. What do you observe and why?

The statistics were automatically gathered by the database during the maintenance window. You can also see this directly from the Automated Maintenance Tasks page in Enterprise Manager. The important thing is that the database automatically gathered the

right statistics and histograms. Depending on your environment, you may see different sample sizes.

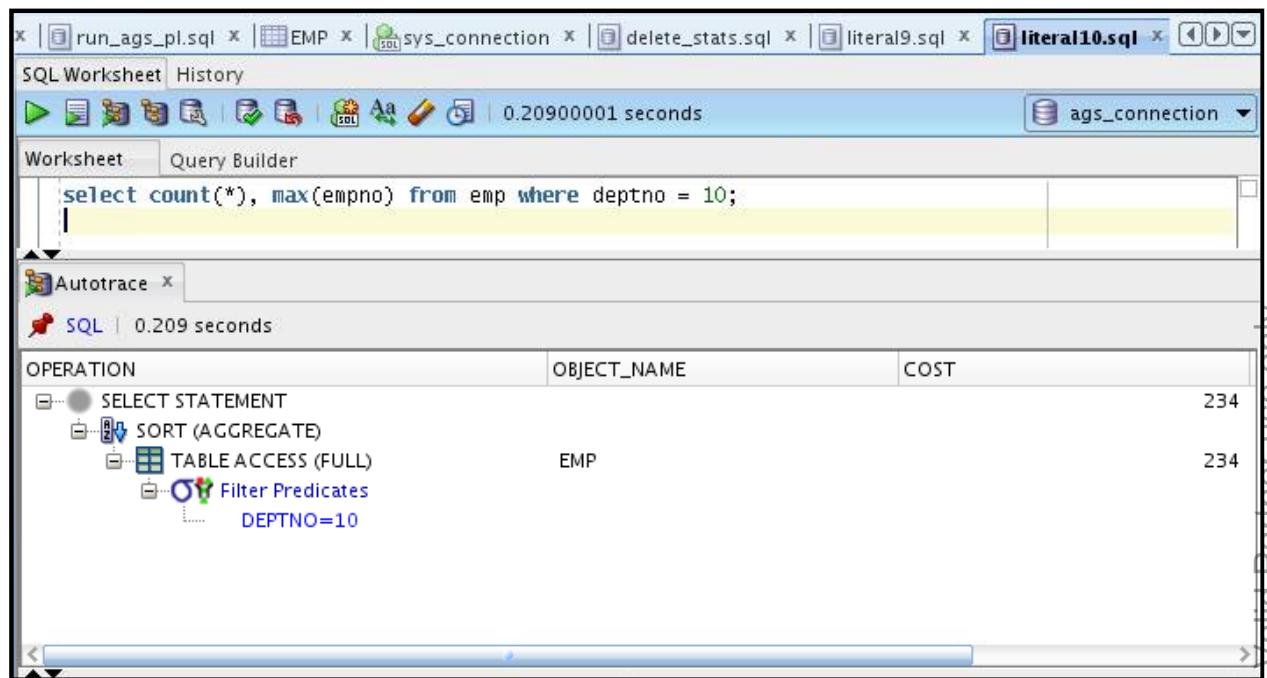
| Name                | Value     |
|---------------------|-----------|
| NUM_ROWS            | 100000    |
| BLOCKS              | 874       |
| AVG_ROW_LEN         | 50        |
| SAMPLE_SIZE         | 100000    |
| LAST_ANALYZED       | 19-JUL-12 |
| LAST_ANALYZED_SINCE | 19-JUL-12 |

| Column Statistics |            |             |              |                                          |                                      |
|-------------------|------------|-------------|--------------|------------------------------------------|--------------------------------------|
| OWNER             | TABLE_NAME | COLUMN_NAME | NUM_DISTINCT | LOW_VALUE                                | HIGH_VALUE                           |
| AGS               | EMP        | EMPNO       | 100000       | C102                                     | C30B                                 |
| AGS               | EMP        | ENAME       | 100000       | 41414142404C424449525450594D564F48545346 | 5A5A5A5958524A594A4F4E555852425A4149 |
| AGS               | EMP        | PHONE       | 100000       | 41414142484541584D4D46564252564A5258494E | 5A5A5A5A4851485344554E564A4F444E454A |
| AGS               | EMP        | DEPTNO      | 11 80        |                                          | C10B                                 |

25. Autotrace literal9.sql and literal10.sql again. What do you observe and why?
- The optimizer can make the right decisions for both statements. This is because of the statistics that were automatically gathered by the database previously.

| OPERATION                     | OBJECT_NAME | COST |
|-------------------------------|-------------|------|
| SELECT STATEMENT              |             | 2    |
| SORT (AGGREGATE)              |             | 2    |
| TABLE ACCESS (BY INDEX ROWID) | EMP         | 1    |
| INDEX (RANGE SCAN)            | EMP_I1      |      |
| Access Predicates             |             |      |
| DEPTNO=9                      |             |      |



26. Close ags\_connection and the file tabs in SQL Developer.
27. From your terminal window, clean up your environment by executing the ags\_cleanup.sh script.

```
$./ags_cleanup.sh
...
SQL>
SQL> alter system set "_enable_automatic_maintenance"=1;
System altered.

SQL>
SQL> exit;
Disconnected ...
$
```

## Practice 11- 4: Using System Statistics (Optional)

### Overview

In this practice, you manipulate system statistics and show that they are important for the optimizer to select the correct execution plans. All the scripts used in this practice are located in the \$HOME/solutions/System\_Stats directory.

### Tasks

1. The sysstats\_setup.sh script located in your \$HOME/solutions/System\_Stats directory was run as part of the class setup. This script creates a user called JFV and some tables that are used throughout this practice. The script also makes sure that object statistics are correctly gathered. The script is listed as follows:

```
#!/bin/bash

cd /home/oracle/solutions/System_Stats

sqlplus / as sysdba @sysstats_setup.sql

set echo on

connect / as sysdba;

drop user jfv cascade;

create user jfv identified by jfv default tablespace users temporary
tablespace temp;

grant connect, resource, dba to jfv;

connect jfv/jfv

drop table t purge;

drop table z purge;

create table t(c number);

insert into t values (1);

commit;

insert into t select * from t;
```

```
/
/
/
/
/
/
/
/
/
/
/
/

commit;

insert into t select * from t;

/
/
/
/
/
/
/
/
/
/

commit;

create table z(d number);

begin
for i in 1..100 loop
 insert into z values (i);
end loop;
commit;
end;
/

create unique index iz on z(d);

execute dbms_stats.gather_table_stats('JFV','T',cascade=>true);

execute dbms_stats.gather_table_stats('JFV','Z',cascade=>true);

exit;
```

2. In a terminal window, execute the `sysstats_start.sh` script to flush the caches, and set `CPUSPEEDNW` to 0 in the system statistics.

```
$ cd /home/oracle/solutions/System_Stats
$./sysstats_start.sh

...
SQL>
SQL> alter system flush shared_pool;

System altered.

SQL>
SQL> alter system flush buffer_cache;

System altered.

SQL>
SQL> execute dbms_stats.delete_system_stats;

PL/SQL procedure successfully completed.

SQL>
SQL> execute DBMS_STATS.SET_SYSTEM_STATS (pname => 'cpuspeednw',
pvalue => 0);

PL/SQL procedure successfully completed.

SQL>
SQL> select sname,pname,pval1 from aux_stats$;

SNAME PNAME PVAL1
----- -----
SYSSTATS_INFO STATUS
SYSSTATS_INFO DSTART
SYSSTATS_INFO DSTOP
SYSSTATS_INFO FLAGS 1
SYSSTATS_MAIN CPUSPEEDNW 0
SYSSTATS_MAIN IOSEEKTIM 10
SYSSTATS_MAIN IOTFRSPEED 4096
SYSSTATS_MAIN SREADTIM
SYSSTATS_MAIN MREADTIM
SYSSTATS_MAIN CPUSPEED
SYSSTATS_MAIN MBRC
```

| SNAME             | PNAME    |
|-------------------|----------|
| PVAL1             |          |
| -----             | -----    |
| SYSSTATS_MAIN     | MAXTHR   |
| SYSSTATS_MAIN     | SLAVETHR |
| 13 rows selected. |          |
| SQL>              |          |
| SQL> exit;        |          |
| ...               |          |
| \$                |          |

3. From your terminal session, connect as the JFV user in a SQL\*Plus session. Then execute the `select_without_sysstats.sql` script, which contains the following statement, and determine how long it takes to execute:

```
select /* Without system stats */ count(*)
from t,z
where t.c=z.d;
```

```
$ sqlplus jfv/jfv

...
SQL> @select_without_sysstats
SQL>
SQL> set timing on
SQL>
SQL> select /* Without system stats */ count(*)
2 from t,z
3 where t.c=z.d;

COUNT(*)

 524288

Elapsed: 00:00:00.23
SQL>
SQL> set timing off
SQL>
SQL>
```

4. Determine the execution plan that was used to execute the previous statement. In addition, determine the optimizer's cost, CPU cost, and I/O cost for the previous execution. Use the `show_latest_exec_plan.sql` script. What do you observe?

- a. The optimizer does not use CPU costing. This is because system statistics were deleted during the first step of this practice. The plan chosen by the optimizer might not be the best one.

```
SQL> @show_latest_exec_plan
SQL> set echo on
SQL>
SQL> select * from table(dbms_xplan.display_cursor);

PLAN_TABLE_OUTPUT

SQL_ID 6avdu58tamzju, child number 0

select /* Without system stats */ count(*) from t,z where t.c=z.d

Plan hash value: 3698032250

| Id | Operation | Name | Rows | Bytes | Cost |

| 0 | SELECT STATEMENT | | | | 134 |
| 1 | SORT AGGREGATE | | | 1 | 6 |

```

```
PLAN_TABLE_OUTPUT

| 2 | NESTED LOOPS | | 524K | 3072K | 134 |
| 3 | TABLE ACCESS FULL| T | 524K | 1536K | 134 |
|* 4 | INDEX UNIQUE SCAN| IZ | 1 | 3 |

```

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

4 - access("T"."C"="Z"."D")
```

Note

```
PLAN_TABLE_OUTPUT


```

```
- cpu costing is off (consider enabling it)
```

```
25 rows selected.
```

```

SQL>
SQL> col operations format a20
SQL> col object_name format a11
SQL> col options format a15
SQL> col cost_cpu_io format a30
SQL>
SQL>
SQL> select operation operations, object_name, options,
2 cost||' -- '||cpu_cost||' -- '||io_cost cost_cpu_io
3 from (select * from v$sql_plan where address in (select address
4 from v$sql
5 where sql_text
like '%system stats%' and
6 sql_text
not like '%connect%'));

OPERATIONS OBJECT_NAME OPTIONS COST_CPU_IO
----- -----
----- -----
SELECT STATEMENT 134 -- --
SORT AGGREGATE -- --
NESTED LOOPS 134 -- -- 134
TABLE ACCESS T FULL 134 -- -- 134
INDEX IZ UNIQUE SCAN -- --
SQL>
SQL>
```

5. How can you ensure that the optimizer finds a better plan during future executions of the same statement? Implement your solution.
- Gather system statistics again. Because you do not have a real workload yet, you can gather system statistics in NOWORKLOAD mode.

```

SQL> connect / as sysdba;
Connected.
SQL> @gather_system_stats
SQL> set echo on
SQL>
SQL> execute DBMS_STATS.GATHER_SYSTEM_STATS(gathering_mode =>
'NOWORKLOAD');

PL/SQL procedure successfully completed.

SQL>
SQL> select sname,pname,pval1 from aux_stats$;

SNAME PNAME PVAL1
----- ----- -----

```

```

SYSSTATS_INFO STATUS
SYSSTATS_INFO DSTART
SYSSTATS_INFO DSTOP
SYSSTATS_INFO FLAGS 1
SYSSTATS_MAIN CPUSPEEDNW 1882.885
SYSSTATS_MAIN IOSEEKTIM 10.506
SYSSTATS_MAIN IOTFRSPEED 4096
SYSSTATS_MAIN SREADTIM
SYSSTATS_MAIN MREADTIM
SYSSTATS_MAIN CPUSPEED
SYSSTATS_MAIN MBRC

SNAME PNAME
PVAL1

SYSSTATS_MAIN MAXTHR
SYSSTATS_MAIN SLAVETHR

13 rows selected.

SQL>
```

6. Before verifying your solution, you should flush the System Global Area (SGA) pools, such as the shared pool and the buffer cache. This is done to prevent the already loaded buffers or SQL plans from affecting the results.

```
SQL> @flush_sga
SQL>
SQL> set echo on
SQL>
SQL> alter system flush shared_pool;

System altered.

SQL>
SQL> alter system flush buffer_cache;

System altered.

SQL>
SQL>
```

7. Connected as the JFV user again, check your solution against the following query in the select\_with\_sysstats.sql script:

```
select /* With system stats */ count(*)
from t,z
where t.c=z.d;
```

What do you observe?

- a. The optimizer can make a better decision because it was able to use meaningful system statistics. You can see that the execution time is now half the value it was previously, and the execution plan now includes CPU costing information.

```
SQL> connect jfv/jfv
Connected.
SQL> @select_with_sysstats
SQL> set timing on
SQL>
SQL> select /* With system stats */ count(*)
 2 from t,z
 3 where t.c=z.d;

 COUNT(*)

 524288

Elapsed: 00:00:00.11
SQL>
SQL> set timing off
SQL>
SQL>
SQL> @show_latest_exec_plan
SQL> set echo on
SQL>
SQL> select * from table(dbms_xplan.display_cursor);

PLAN_TABLE_OUTPUT

SQL_ID 2x55txn3742by, child number 0

select /* With system stats */ count(*) from t,z where t.c=z.d

Plan hash value: 2407521827

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time
|---|

| 0 | SELECT STATEMENT | | | | 240 (100)|
```

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

```

| 1 | SORT AGGREGATE | | | 1 | 6 | |
|-----|
PLAN_TABLE_OUTPUT

|* 2 | HASH JOIN | | | 524K| 3072K| 240 (4) |
00:00:04 |
| 3 | INDEX FULL SCAN | IZ | | 100 | 300 | 1 (0) |
00:00:01 |
| 4 | TABLE ACCESS FULL| T | | 524K| 1536K| 235 (2) |
00:00:03 |
|-----|
Predicate Information (identified by operation id):

2 - access("T"."C"="Z"."D")

21 rows selected.

SQL>
SQL> col operations format a20
SQL> col object_name format a11
SQL> col options format a15
SQL> col cost_cpu_io format a30
SQL>
SQL>
SQL> select operation operations, object_name, options,
2 cost||' -- '|cpu_cost||' -- '|io_cost cost_cpu_io
3 from (select * from v$sql_plan
4 where address in (select address from v$sql
5 where sql_text like '%system stats%'
6 and sql_text not like '%connect%'));

```

| OPERATIONS       | OBJECT_NAME | OPTIONS   | COST_CPU_IO             |
|------------------|-------------|-----------|-------------------------|
| SELECT STATEMENT |             |           | 240 -- --               |
| SORT             |             | AGGREGATE | -- --                   |
| HASH JOIN        |             |           | 240 -- 149111940 -- 234 |
| INDEX            | IZ          | FULL SCAN | 1 -- 27121 -- 1         |
| TABLE ACCESS     | T           | FULL      | 237 -- 84867339 -- 233  |

```

SQL>

```

8. Exit your SQL\*Plus session and clean up your environment for this practice by executing the `systats_cleanup.sh` script.

```
SQL> exit
...
$./sysstats_cleanup.sh
...
SQL>
SQL> drop user jfv cascade;

User dropped.

SQL>
SQL> alter system flush shared_pool;

System altered.

SQL>
SQL> exit;
```

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Error : You are not a Valid Partner use only

## **Practices for Lesson 12: Using Bind Variables**

**Chapter 12**

## Practices for Lesson 12: Overview

---

### Practices Overview

In these practices, you examine the behavior of adaptive cursor sharing and the effect of various settings of the `CURSOR_SHARING` parameter on execution plans.

## Practice 12-1: Understanding Adaptive Cursor Sharing

### Overview

In this practice, you experiment with bind variable peeking and adaptive cursor sharing.

### Tasks

1. The `acs_setup.sh` script was executed as part of the class setup to set up the environment used for this practice. This script is in your `$HOME/labs/solutions/Adaptive_Cursor_Sharing` directory and is listed as follows:

```
set echo on

drop user acs cascade;

create user acs identified by acs default tablespace users temporary
tablespace temp;

grant dba, connect to acs;

connect acs/acs

drop table emp purge;

create table emp
(
empno number,
ename varchar2(20),
phone varchar2(20),
deptno number
);

insert into emp
with tdata as
 (select rownum empno
 from all_objects
 where rownum <= 1000)
select rownum,
 dbms_random.string ('u', 20),
 dbms_random.string ('u', 20),
 case
 when rownum/100000 <= 0.001 then mod(rownum, 10)
 else 10
 end
 from tdata a, tdata b
 where rownum <= 100000;
```

```

create index emp_i1 on emp(deptno);

exec dbms_stats.gather_table_stats(null, 'EMP', METHOD_OPT => 'FOR
COLUMNS DEPTNO SIZE 10', CASCADE => TRUE);

alter system flush shared_pool;
exit;

```

- Change directories to \$HOME/labs/solutions/Adaptive\_Cursor\_Sharing. In your terminal session, connect to the SQL\*Plus session as the ACS user. Ensure that you stay connected to the same SQL\*Plus session until the end of this practice. After you are connected, identify the columns of the EMP table that have histograms. Flush the shared pool before you start.

Only the DEPTNO column has a 10-buckets histogram.

```

$ cd $HOME/labs/solutions/Adaptive_Cursor_Sharing
$ sqlplus acs/acs
...
SQL> alter system flush shared_pool;

System altered.

SQL> @check_emp_histogram
SQL>
SQL> select column_name, histogram, num_buckets
 2 from user_tab_columns
 3 where table_name='EMP';

COLUMN_NAME HISTOGRAM NUM_BUCKETS
----- -----
EMPNO NONE
ENAME NONE
PHONE NONE
DEPTNO TOP-FREQUENCY 10
SQL>

```

- Determine the distribution of all the distinct values found in the DEPTNO column of the EMP table. What do you find?

Values distribution is uniform for all (0.01%), except for value 10 (99.9%). This is typical of what is called data skew.

```

SQL> @show_deptno_distribution
SQL> set echo on
SQL>

```

```

SQL> select deptno, count(*) cnt_per_deptno, (count(*)*100)/nr
deptno_percent
 2 from emp, (select max(empno) nr
 3 from emp)
 4 group by deptno, nr
 5 order by deptno;

 DEPTNO CNT_PER_DEPTNO DEPTNO_PERCENT
----- -----
 0 10 .01
 1 10 .01
 2 10 .01
 3 10 .01
 4 10 .01
 5 10 .01
 6 10 .01
 7 10 .01
 8 10 .01
 9 10 .01
 10 99900 99.9

11 rows selected.

```

4. Before you study the adaptive cursor-sharing feature, disable its functionality by setting the OPTIMIZER\_FEATURES\_ENABLE session parameter back to 10.2.0.1. After this is done, ensure that you execute the following command in your SQL\*Plus session: set lines 200 pages 10000. This is used in the practice to print the execution plans correctly.

```

SQL> alter session set optimizer_features_enable="10.2.0.1";
Session altered.

SQL> set lines 200 pages 10000
SQL>

```

5. Determine the execution plan for the following statement:

```

select /*ACS_L9*/ count(*), max(empno)
 from emp
 where deptno = 9;

```

This statement is in the select\_deptno\_literal\_9.sql file.

What do you notice and why?

- a. The optimizer uses an index range scan because value 9 is very selective.

```
SQL> @select_deptno_literal_9
```

```
SQL> set echo on
SQL>
SQL> select /*ACS_L9*/ count(*), max(empno)
 2 from emp
 3 where deptno = 9;

 COUNT(*) MAX(EMPNO)

 10 99

SQL>
SQL> @show_latest_exec_plan.sql
SQL> set echo on
SQL>
SQL> select * from table(dbms_xplan.display_cursor(null,null,'TYPICAL
+PEEKED_BINDS'));

PLAN_TABLE_OUTPUT

SQL_ID 64ngy4j55d1z5, child number 0

select /*ACS_L9*/ count(*), max(empno) from emp where deptno = 9

Plan hash value: 3184478295

| Id | Operation | Name | Rows | Bytes | Cost
(%CPU)| Time |

| 0 | SELECT STATEMENT | | | | 2
(100)| |
| 1 | SORT AGGREGATE | | | 16 | 2
| |
| 2 | TABLE ACCESS BY INDEX ROWID| EMP | 19 | 304 | 2
(0)| 00:00:01 |
|* 3 | INDEX RANGE SCAN | EMP_I1 | 19 | | 1
(0)| 00:00:01 |

Predicate Information (identified by operation id):

3 - access ("DEPTNO"=9)
```

```
20 rows selected.
```

```
SQL>
```

6. Determine the execution plan for the following statement:

```
select /*ACS_L10*/ count(*), max(empno)
from emp
where deptno = 10;
```

This statement is in the `select_deptno_literal_10.sql` file.

What do you notice and why?

- a. The optimizer uses a full table scan because value 10 is not a selective value.

```
SQL> @select_deptno_literal_10
SQL> set echo on
SQL>
SQL> select /*ACS_L10*/ count(*), max(empno)
 2 from emp
 3 where deptno = 10;

 COUNT(*) MAX(EMPNO)

 99900 100000

SQL>
SQL> @show_latest_exec_plan
SQL> set echo on
SQL>
SQL> select * from table(dbms_xplan.display_cursor(null,null,'TYPICAL
+PEEKED_BINDS'));

PLAN_TABLE_OUTPUT

SQL_ID 3232j5gkp2u5h, child number 0

select /*ACS_L10*/ count(*), max(empno) from emp where deptno = 10

Plan hash value: 2083865914

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time
|----|----|----|----|----|----|----|
```

```

| 0 | SELECT STATEMENT | | | | 239 (100) |
| 1 | SORT AGGREGATE | | | 1 | 16 | |
|* 2 | TABLE ACCESS FULL| EMP | 95000 | 1484K| 239 (1) |
00:00:03 |

Predicate Information (identified by operation id):

2 - filter("DEPTNO"=10)

19 rows selected.

SQL>

```

7. Define a bind variable called DEPTNO in your SQL\*Plus session, set it to value 9, execute the following query, and determine its execution plan:

```

select /*ACS_1*/ count(*), max(empno)
from emp
where deptno = :deptno;

```

This statement is in the select\_deptno\_bind.sql file.

What do you notice and why?

Because the optimizer uses bind peeking the first time you execute a statement with a bind variable and because, for this first execution, value 9 is used, the execution plan with index access is used.

```

SQL> variable deptno number;
SQL> exec :deptno := 9

PL/SQL procedure successfully completed.

SQL> @select_deptno_bind
SQL> set echo on
SQL>
SQL> select /*ACS_1*/ count(*), max(empno)
2 from emp
3 where deptno = :deptno;

COUNT(*) MAX(EMPNO)

 10 99

```

```
SQL>
SQL> @show_latest_exec_plan
SQL> set echo on
SQL>
SQL> select * from table(dbms_xplan.display_cursor(null,null,'TYPICAL
+PEEKED_BINDS'));

PLAN_TABLE_OUTPUT

SQL_ID 272gr4hapc9w1, child number 0

select /*ACS_1*/ count(*), max(empno) from emp where deptno = :deptno

Plan hash value: 3184478295

| Id | Operation | Name | Rows | Bytes | Cost |
| (%CPU)| Time | | | | |

| 0 | SELECT STATEMENT | | | | | 2 |
| (100)| | | | | |
| 1 | SORT AGGREGATE | | | 1 | 16 |
| | | | | | |
| 2 | TABLE ACCESS BY INDEX ROWID| EMP | 19 | 304 | 2 |
| (0)| 00:00:01 | | | | |
|* 3 | INDEX RANGE SCAN | EMP_I1 | 19 | | 1 |
| (0)| 00:00:01 | | | | |

Peeked Binds (identified by position):

1 - :DEPTNO (NUMBER): 9

Predicate Information (identified by operation id):

3 - access ("DEPTNO"=:DEPTNO)

25 rows selected.

SQL>
```

```
SQL>
```

8. Determine the execution statistics in terms of child cursors, executions, and buffer gets for the previously executed statement. What do you observe?

- a. In V\$SQL, only one child cursor exists, and it has been executed only once (first time ever in this case). Also, the number of buffer gets is small due to the efficient access path that was used.

```
SQL> @show_latest_exec_stats
SQL> set echo on
SQL>
SQL> select child_number, executions, buffer_gets
 2 from v$sql
 3 where sql_text like 'select /*ACS_1%';

CHILD_NUMBER EXECUTIONS BUFFER_GETS
----- ----- -----
 0 1 3

SQL>
```

9. Perform steps 7 and 8 again, but this time using 10 as the bind value for DEPTNO. What do you observe and why?

- a. The execution plan is identical. The index path is used, although value 10 is not selective. This is because bind peeking operates only the first time you execute your statement. Notice that the PEEK\_BIND value is still 9. Looking at V\$SQL, you can clearly see that there is still only one child cursor associated with your statement. However, this time, the number of buffer gets was raised significantly due to the number of accesses required to retrieve all the rows from first the index, and then the table.

```
SQL> exec :deptno := 10
PL/SQL procedure successfully completed.
SQL> @select_deptno_bind
SQL> set echo on
SQL>
SQL> select /*ACS_1*/ count(*), max(empno)
 2 from emp
 3 where deptno = :deptno;

COUNT(*) MAX(EMPNO)
----- -----
 99900 100000

SQL>
SQL> @show_latest_exec_plan
SQL> set echo on
```

```
SQL>
SQL> select * from table(dbms_xplan.display_cursor(null,null,'TYPICAL
+PEEKED_BINDS'));

PLAN_TABLE_OUTPUT

SQL_ID 272gr4hapc9w1, child number 0

select /*ACS_1*/ count(*), max(empno) from emp where deptno = :deptno

Plan hash value: 3184478295

| Id | Operation | Name | Rows | Bytes | Cost |
| (%CPU)| Time | | | | |

| 0 | SELECT STATEMENT | | | | | 2 |
| (100)| | | | | |
| 1 | SORT AGGREGATE | | | 1 | 16 |
| | | | | | |
| 2 | TABLE ACCESS BY INDEX ROWID| EMP | 19 | 304 | 2 |
| (0) | 00:00:01 | | | | |
|* 3 | INDEX RANGE SCAN | EMP_I1 | 19 | | 1 |
| (0) | 00:00:01 | | | | |


```

Peeked Binds (identified by position):

```
1 - :DEPTNO (NUMBER) : 9
```

Predicate Information (identified by operation id):

```
3 - access ("DEPTNO"=:DEPTNO)
```

25 rows selected.

```
SQL>
SQL> @show_latest_exec_stats
SQL> set echo on
SQL>
```

```
SQL> select child_number, executions, buffer_gets
 2 from v$sql
 3 where sql_text like 'select /*ACS_1%';

CHILD_NUMBER EXECUTIONS BUFFER_GETS

 0 2 957

SQL>
```

10. Before the next step, flush your shared pool to make sure that you clear all the cursor information.

```
SQL> alter system flush shared_pool;

System altered.

SQL>
```

11. Perform step 9 again. This time, set the bind variable to 10. What do you observe and why?

- a. The execution plan is a full table scan because you used value 10 as your first bind value. You can see this in the peeked binds. There is only one child cursor that is created so far to handle your statement.

```
SQL> exec :deptno := 10

PL/SQL procedure successfully completed.

SQL> @select_deptno_bind
SQL> set echo on
SQL>
SQL> select /*ACS_1*/ count(*), max(empno)
 2 from emp
 3 where deptno = :deptno;

COUNT(*) MAX(EMPNO)

 99900 100000

SQL>
SQL> @show_latest_exec_plan
SQL> set echo on
SQL>
```

```
SQL> select * from table(dbms_xplan.display_cursor(null,null,'TYPICAL
+PEEKED_BINDS'));

PLAN_TABLE_OUTPUT

SQL_ID 272gr4hapc9w1, child number 0

select /*ACS_1*/ count(*), max(empno) from emp where deptno = :deptno

Plan hash value: 2083865914

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time
|---|---|---|---|---|---|---|
| 0 | SELECT STATEMENT | | | | | 239 (100) |
| 1 | SORT AGGREGATE | | | | | 16 |
| * 2 | TABLE ACCESS FULL| EMP | 95000 | 1484K| 239 (1) |
| 00:00:03 | |

|---|---|---|---|---|---|---|
| 1 - :DEPTNO (NUMBER): 10
| Predicate Information (identified by operation id):
|-----|
| 2 - filter("DEPTNO"=:DEPTNO)
|-----|
24 rows selected.

SQL>
SQL>
SQL> @show_latest_exec_stats
SQL> set echo on
SQL>
SQL> select child_number, executions, buffer_gets
2 from v$sql
```

```

3 where sql_text like 'select /*ACS_1%';

CHILD_NUMBER EXECUTIONS BUFFER_GETS

0 1 876

SQL>
```

12. Perform step 9 again, but this time, use 9 as your bind value. What do you observe and why?
- Although value 9 is very selective, a full table scan is still used. This is because the second time you execute your statement, bind peeking is not done. So you continue to use the same child cursor.

```

SQL> exec :deptno := 9

PL/SQL procedure successfully completed.

SQL> @select_deptno_bind
SQL> set echo on
SQL>
SQL> select /*ACS_1*/ count(*), max(empno)
2 from emp
3 where deptno = :deptno;

COUNT(*) MAX(EMPNO)

10 99

SQL>
SQL> @show_latest_exec_plan
SQL> set echo on
SQL>
SQL> select * from table(dbms_xplan.display_cursor(null,null,'TYPICAL
+PEEKED_BINDS'));

PLAN_TABLE_OUTPUT

SQL_ID 272gr4hapc9w1, child number 0

select /*ACS_1*/ count(*), max(empno) from emp where deptno = :deptno

Plan hash value: 2083865914

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time

```

```

| 0 | SELECT STATEMENT | | | | | 239 (100) |
| 1 | SORT AGGREGATE | | | | |
|* 2 | TABLE ACCESS FULL | EMP | 95000 | 1484K | 239 (1) |
00:00:03 |

Peeked Binds (identified by position):

1 - :DEPTNO (NUMBER): 10

Predicate Information (identified by operation id):

2 - filter("DEPTNO"=:DEPTNO)

24 rows selected.

SQL>
SQL> @show_latest_exec_stats
SQL> set echo on
SQL>
SQL> select child_number, executions, buffer_gets
 2 from v$sql
 3 where sql_text like 'select /*ACS_1%';
-----+
CHILD_NUMBER EXECUTIONS BUFFER_GETS
-----+
 0 2 1695
SQL>
```

13. Before the next step, reset your session to use adaptive cursor sharing and ensure that you flush your shared pool again.

```

SQL> alter session set optimizer_features_enable="12.1.0.1";
Session altered.

SQL> alter system flush shared_pool;
System altered.

SQL>
```

14. Perform step 12 again. What do you observe and why?

- a. Because this is the first time you execute the statement, bind peeking is used, and because value 9 is very selective, the index path is used. Only one child cursor is used to handle this statement.

```
SQL> exec :deptno := 9

PL/SQL procedure successfully completed.

SQL> @select_deptno_bind
SQL> set echo on
SQL>
SQL> select /*ACS_1*/ count(*), max(empno)
 2 from emp
 3 where deptno = :deptno;

 COUNT(*) MAX(EMPNO)

 10 99

SQL>
SQL> @show_latest_exec_plan
SQL> set echo on
SQL>
SQL> select * from table(dbms_xplan.display_cursor(null,null,'TYPICAL
+PEEKED_BINDS'));

PLAN_TABLE_OUTPUT

SQL_ID 272gr4hapc9w1, child number 0

select /*ACS_1*/ count(*), max(empno) from emp where deptno = :deptno

Plan hash value: 3184478295

| Id | Operation | Name | Rows | Bytes | Cost |
| (%CPU)| Time | | | | |

| 0 | SELECT STATEMENT | | | | 2 |
| (100)| | | | | |
| 1 | SORT AGGREGATE | | 1 | 16 | |
| | | | | | |
| 2 | TABLE ACCESS BY INDEX ROWID| EMP | 1 | 16 | 2 |
| (0)| 00:00:01 | | | | |

```

```
|* 3 | INDEX RANGE SCAN | EMP_I1 | 1 | | 1
(0) | 00:00:01 |

Peeked Binds (identified by position):

1 - :DEPTNO (NUMBER): 9

Predicate Information (identified by operation id):

3 - access ("DEPTNO"=:DEPTNO)

25 rows selected.

SQL>
SQL> @show_latest_exec_stats
SQL> set echo on
SQL>
SQL> select child_number, executions, buffer_gets
 2 from v$sql
 3 where sql_text like 'select /*ACS_1%';
-----+
CHILD_NUMBER EXECUTIONS BUFFER_GETS
-----+
 0 1 61
-----+
SQL>
```

15. Perform step 14 again, but this time using value 10 as your bind value. What do you observe and why?
- Although value 10 is not selective, the same index path as in the previous step is used. Only one child cursor is currently needed to represent your statement.

```
SQL> exec :deptno := 10
PL/SQL procedure successfully completed.

SQL> @select_deptno_bind
SQL> set echo on
SQL>
SQL> select /*ACS_1*/ count(*), max(empno)
 2 from emp
 3 where deptno = :deptno;
```

```
COUNT(*) MAX(EMPNO)

99900 100000

SQL>
SQL> @show_latest_exec_plan
SQL> set echo on
SQL>
SQL> select * from table(dbms_xplan.display_cursor(null,null,'TYPICAL
+PEEKED_BINDS'));

PLAN_TABLE_OUTPUT

SQL_ID 272gr4hapc9w1, child number 0

select /*ACS_1*/ count(*), max(empno) from emp where deptno = :deptno

Plan hash value: 3184478295

| Id | Operation | Name | Rows | Bytes | Cost
(%CPU)| Time |

| 0 | SELECT STATEMENT | | | | | 2
(100)| |
| 1 | SORT AGGREGATE | | | 1 | 16 |
| 2 | TABLE ACCESS BY INDEX ROWID| EMP | | 16 | 2
(0) | 00:00:01 |
|* 3 | INDEX RANGE SCAN | EMP_I1 | | | | 1
(0) | 00:00:01 |

Peeked Binds (identified by position):

1 - :DEPTNO (NUMBER): 9

Predicate Information (identified by operation id):

3 - access("DEPTNO"=:DEPTNO)
```

```
25 rows selected.

SQL>
SQL> @show_latest_exec_stats
SQL> set echo on
SQL>
SQL> select child_number, executions, buffer_gets
 2 from v$sql
 3 where sql_text like 'select /*ACS_1%';

CHILD_NUMBER EXECUTIONS BUFFER_GETS

 0 2 1015
```

16. Perform step 15 again. What do you observe and why?

- Because you now use adaptive cursor sharing, the system realizes that you benefit from another child cursor for handling your statement. This time, a full table access path is used to better handle your statement.

```
SQL> exec :deptno := 10

PL/SQL procedure successfully completed.

SQL> @select_deptno_bind
SQL> set echo on
SQL>
SQL> select /*ACS_1*/ count(*), max(empno)
 2 from emp
 3 where deptno = :deptno;

COUNT(*) MAX(EMPNO)

 99900 100000

SQL>
SQL> @show_latest_exec_plan
SQL> set echo on
SQL>
SQL> select * from table(dbms_xplan.display_cursor(null,null,'TYPICAL
+PEEKED_BINDS'));

PLAN_TABLE_OUTPUT
```

```

SQL_ID 272gr4hapc9w1, child number 1

select /*ACS_1*/ count(*), max(empno) from emp where deptno = :deptno

Plan hash value: 2083865914

-----| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time
-----| 0 | SELECT STATEMENT | | | | | 239 (100) |
-----| 1 | SORT AGGREGATE | | | 1 | 16 | |
-----|* 2 | TABLE ACCESS FULL | EMP | 95000 | 1484K| 239 (1) |
00:00:03 |
-----| Peaked Binds (identified by position):
-----| 1 - :DEPTNO (NUMBER): 10
-----| Predicate Information (identified by operation id):
-----| 2 - filter("DEPTNO"=:DEPTNO)
-----| 24 rows selected.

SQL>
SQL> @show_latest_exec_stats
SQL> set echo on
SQL>
SQL> select child_number, executions, buffer_gets
2 from v$sql
3 where sql_text like 'select /*ACS_1%';
-----| CHILD_NUMBER EXECUTIONS BUFFER_GETS
-----| 0 2 1015
-----|

```

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

|      |   |     |
|------|---|-----|
| 1    | 1 | 806 |
| SQL> |   |     |

17. Flush the shared pool to clean up your environment, and then exit your SQL\*Plus session.

```
SQL> alter system flush shared_pool;
System altered.
SQL> exit;
```

## Practice 12-2: Understanding CURSOR\_SHARING (Optional)

In this practice, you investigate the use of the CURSOR\_SHARING initialization parameter.

1. You can find all the necessary scripts for this practice in your \$HOME/labs/solutions/Cursor\_Sharing directory. The environment for this practice has been set up with the class setup, using the cs\_setup.sh script. This script created a new user called CS and the EMP table that are used throughout this practice. The script is listed as follows:

```
#!/bin/bash

cd /home/oracle/labs/solutions/Cursor_Sharing

sqlplus / as sysdba @cs_setup.sql

set echo on

drop user cs cascade;

create user cs identified by cs default tablespace users temporary
tablespace temp;

grant dba, connect to cs;

connect cs/cs

drop table emp purge;

create table emp
(
 empno number,
 ename varchar2(20),
 phone varchar2(20),
 deptno number
);

insert into emp
 with tdata as
 (select rownum empno
 from all_objects
 where rownum <= 1000)
 select rownum,
 dbms_random.string ('u', 20),
 dbms_random.string ('u', 20),
```

```
case
 when rownum/100000 <= 0.001 then mod(rownum, 10)
 else 10
 end
from tdata a, tdata b
where rownum <= 100000;

create index emp_i1 on emp(deptno);

execute dbms_stats.gather_table_stats(null, 'EMP', cascade => true);

--alter system flush shared_pool;

--connect / as sysdba

--shutdown immediate;

--startup;

exit;
```

2. In a terminal session, change directory to the \$HOME/labs/solutions/Cursor\_Sharing directory. Connect as the CS user in a SQL\*Plus session and stay connected to that session until the end of this practice. For formatting reasons, after you have connected in the SQL\*Plus session, execute the following command:

```
set linesize 200 pagesize 1000
```

```
$ cd $HOME/labs/solutions/Cursor_Sharing
$ sqlplus cs/cs
...
Connected to: ...
SQL> set linesize 200 pagesize 1000
SQL>
```

3. Check the existence of histograms on the columns of the EMP table by using the check\_emp\_histogram.sql script, and then determine the data distribution in the DEPTNO column of the EMP table with the statement in the show\_deptno\_distribution.sql file. What do you observe?
  - a. Currently, there are no histograms created on the columns of the EMP table. Also, it is clear that you have data skew in the DEPTNO column. Value 10 repeats most of the time (99.9%), whereas all other values repeat only 0.01%.

```
SQL> @check_emp_histogram
SQL>
SQL> select column_name, histogram, num_buckets
 2 from user_tab_columns
 3 where table_name='EMP';

COLUMN_NAME HISTOGRAM NUM_BUCKETS
----- -----
EMPNO NONE 1
ENAME NONE 1
PHONE NONE 1
DEPTNO NONE 1

SQL>
SQL> @show_deptno_distribution
SQL> set echo on
SQL>
SQL> select deptno, count(*) cnt_per_deptno, (count(*)*100)/nr
deptno_percent
 2 from emp, (select max(empno) nr
 3 from emp)
 4 group by deptno, nr
 5 order by deptno;

DEPTNO CNT_PER_DEPTNO DEPTNO_PERCENT
----- -----
 0 10 .01
 1 10 .01
 2 10 .01
 3 10 .01
 4 10 .01
 5 10 .01
 6 10 .01
 7 10 .01
 8 10 .01
 9 10 .01
 10 99900 99.9

11 rows selected.

SQL>
```

4. Before you continue, ensure that you flush your shared pool.

```
SQL> alter system flush shared_pool;

System altered.

SQL>
```

5. How would you force your SQL\*Plus session to automatically replace statement literals with bind variables to make sure that the same cursor is used independently of the literal values?

```
SQL> alter session set cursor_sharing = force;

Session altered.

SQL>
```

6. From the same SQL\*Plus session, execute the following two queries. Then determine how many cursors are generated to handle these two statements and what execution plans were used. What do you observe and why?

```
select /*CS*/ count(*), max(empno) from emp where deptno = 9;
select /*CS*/ count(*), max(empno) from emp where deptno = 10;
```

- a. Because of the previous step, literal values are replaced with bind variables. The FORCE option forces the system to share only one child cursor in this case and use the exact same execution plan (index range scan).

```
SQL> @select_deptno_literal_9
SQL> set echo on
SQL>
SQL> select /*CS*/ count(*), max(empno) from emp where deptno = 9;

COUNT(*) MAX(EMPNO)

 10 99

SQL>
SQL> @select_deptno_literal_10
SQL> set echo on
SQL>
SQL> select /*CS*/ count(*), max(empno) from emp where deptno = 10;

COUNT(*) MAX(EMPNO)

 99900 100000
```

SQL&gt;

- b. Now execute the show\_latest\_cursors.sql file to check the latest cursors.

```
SQL> @show_latest_cursors
SQL> set echo on
SQL>
SQL> col sql_text format a70
SQL>
SQL> select sql_text,hash_value
 2 from v$sql
 3 where sql_text like '%select /*CS%';
SQL_TEXT

HASH_VALUE

select /*CS*/ count(*), max(empno) from emp where deptno =
:"SYS_B_0"
3434097775
```

- c. Execute the show\_latest\_exec\_plans.sql file to check the latest execution plans.

SQL&gt; @show\_latest\_exec\_plans

```

SQL> @show_latest_cursors
SQL> set echo on
SQL>
SQL> col sql_text format a70
SQL>
SQL> select sql_text,hash_value
 2 from v$sql
 3 where sql_text like '%select /*CS%';
SQL_TEXT HASH_VALUE

select /*CS*/ count(*), max(empno) from emp where deptno = :"SYS_B_0" 3434097775

SQL> @show_latest_exec_plans
SQL> set echo on
SQL>
SQL> col child_number Heading 'CHILD|NUMBER'
SQL> col object_name format a6
SQL> col operation format a16
SQL> col options format a15
SQL>
SQL> select address,hash_value,child_number, operation,options,object_name
 2 from v$sql_plan
 3 where (address,hash_value) in
 4 (select address,hash_value
 5 from v$sql
 6 where sql_text like '%select /*CS%'));

 CHILD
ADDRESS HASH_VALUE NUMBER OPERATION OPTIONS OBJECT

000000006BADDFF0 3434097775 0 SELECT STATEMENT
000000006BADDFF0 3434097775 0 SORT AGGREGATE
000000006BADDFF0 3434097775 0 TABLE ACCESS BY INDEX ROWID EMP
 BATCHED

000000006BADDFF0 3434097775 0 INDEX RANGE SCAN EMP_I1

```

7. Ensure that you create a 10-bucket histogram on the DEPTNO column of the EMP table.

```
SQL> exec dbms_stats.gather_table_stats(null, 'EMP', METHOD_OPT =>
'FOR COLUMNS DEPTNO SIZE 10', CASCADE => TRUE);
```

PL/SQL procedure successfully completed.

```
SQL> @check_emp_histogram
SQL> set echo on
SQL>
SQL> select column_name, histogram, num_buckets
 2 from user_tab_columns
 3 where table_name='EMP';
```

| COLUMN_NAME | HISTOGRAM     | NUM_BUCKETS |
|-------------|---------------|-------------|
| EMPNO       | NONE          | 1           |
| ENAME       | NONE          | 1           |
| PHONE       | NONE          | 1           |
| DEPTNO      | TOP-FREQUENCY | 10          |

```
SQL>
```

8. Before you continue, ensure that you flush your shared pool.

```
SQL> alter system flush shared_pool;

System altered.

SQL>
```

9. Perform step 6 again. What do you observe and why?

- a. Although you captured a histogram for the DEPTNO column that shows data skew, the system continues to share only one child cursor to handle both statements. This behavior is due to the FORCE option for the CURSOR\_SHARING initialization parameter.

```
SQL> @select_deptno_literal_9
SQL> set echo on
SQL>
SQL> select /*CS*/ count(*), max(empno) from emp where deptno = 9;

COUNT(*) MAX(EMPNO)

 10 99

SQL>
SQL> @select_deptno_literal_10
SQL> set echo on
SQL>
SQL> select /*CS*/ count(*), max(empno) from emp where deptno = 10;

COUNT(*) MAX(EMPNO)

 99900 100000

SQL>
SQL> @show_latest_cursors
SQL> set echo on
SQL>
SQL> col sql_text format a70
SQL>
SQL> select sql_text,hash_value
2 from v$sql
3 where sql_text like '%select /*CS%';
```

```
SQL_TEXT

HASH_VALUE

select /*CS*/ count(*) , max(empno) from emp where deptno = :"SYS_B_0"
3434097775
```

10. Flush the shared pool script to clean up your environment for this practice. Then exit your SQL\*Plus session.

```
SQL> alter system flush shared_pool;
System altered.

SQL> exit
Disconnected ...

$
```

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Error : You are not a Valid Partner use only

# **Practices for Lesson 13: SQL Plan Management**

**Chapter 13**

## Practices for Lesson 13: Overview

---

### Practices Overview

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:

- Plan Capture
- Plan Selection
- Plan Verification

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

## Practice 13-1: Using SQL Plan Management (SPM)

### Overview

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

### Tasks

1. Before you start this practice, change directories to the \$HOME/labs/solutions/SPM directory. Flush the shared pool to be sure that there is no interference from previous practices.

```
$ cd /home/oracle/labs/solutions/SPM
$ sqlplus / as sysdba
SQL> alter system flush shared_pool;

System altered.

SQL> exit;
```

### 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 enable automatic plan capture so that the SPM repository is automatically populated for any repeatable SQL statement. Enable 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.

```
$ sqlplus spm/spm

-- Execute the below Statement

SQL> show parameter baseline

NAME TYPE VALUE

optimizer_capture_sql_plan_baselines boolean FALSE
optimizer_use_sql_plan_baselines boolean TRUE

-- Execute the below Statement

SQL> alter session set optimizer_capture_sql_plan_baselines =
TRUE;

Session altered.
```

```
SQL>
```

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

```
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> set pagesize 30
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
```

5. Execute the `query1.sql` script again.

```
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> set pagesize 30
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 SQL_6fe28d438dfc352f
select /*LOAD_AUTO*/ * from sh.sales
where quantity_sold > 40 order by prod_id
SQL_PLAN_6zsnd8f6zsd9g54bc8843 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.

This causes the optimizer to create a different plan for the SQL statement execution.

```
SQL> alter session set optimizer_mode = first_rows;
```

```
Session altered.
```

```
-- Execute the query1.sql

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>
```

- a. Because the SQL statement has 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> set pagesize 30
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 SQL_6fe28d438dfc352f
select /*LOAD_AUTO*/ * from sh.sales
where quantity_sold > 40 order by prod_id
SQL_PLAN_6zsnd8f6zsd9g11df68d0 AUTO-CAPTURE YES NO NO YES

8.0622E+18 SQL_6fe28d438dfc352f
select /*LOAD_AUTO*/ * from sh.sales
where quantity_sold > 40 order by prod_id
SQL_PLAN_6zsnd8f6zsd9g54bc8843 AUTO-CAPTURE YES YES NO YES

SQL>
```

- b. 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 automatic capture of plan baselines.

```
-- Execute the below Statement

SQL> alter session set optimizer_mode = all_rows;

Session altered.

-- Execute the below Statement

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('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.

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.

-- Execute the query2.sql Statement

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.

```
-- Execute the below Statement

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

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>

-- Execute the query2.sql Statement

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.

```
-- Execute the below Statement

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.

```
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 the SQL statements that 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>

```

16. Populate the plan baseline repository with the plans found in SPM\_STS. Use the populate\_baseline.sql script:

```

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 whether 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> set pagesize 30
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 SQL_a8632bd857a4a25e

```

```
select /*LOAD_STS*/ * from sh.sales
where quantity sold > 40 order by prod_id
SQL_PLAN_ahstbv1bu98ky11df68d0 MANUAL-LOAD YES YES NO YES

1.2134E+19 SQL_a8632bd857a4a25e
select /*LOAD_STS*/ * from sh.sales
where quantity sold > 40 order by prod_id
SQL_PLAN_ahstbv1bu98ky54bc8843 MANUAL-LOAD YES YES NO YES
```

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(
-> '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>
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>
```

```
-- Execute the query3.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.

```
-- Execute the below Statement

SQL> alter session set optimizer_mode = first_rows;

Session altered.

-- Execute the explain_query3.sql

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;

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 |

12 rows selected.

SQL>

SQL>

-- Execute the query3.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.

-- Execute the below Statement

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>
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 whether the baselines were loaded. (Use `check_baselines3.sql`.)

```
SQL> @check_baselines3.sql
SQL> set echo on
SQL> set pagesize 30
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 SQL_f6cb7f742ef93547<br>select /*LOAD_CC*/ * from sh.sales<br>where quantity_sold > 40 order by prod_id<br>SQL_PLAN_gdkvzfhrkgkda711df68d0 | MANUAL-LOAD | YES | YES | NO  | YES |
| 1.7783E+19 SQL_f6cb7f742ef93547<br>select /*LOAD_CC*/ * from sh.sales<br>where quantity_sold > 40 order by prod_id<br>SQL_PLAN_gdkvzfhrkgkda754bc8843 | MANUAL-LOAD | YES | YES | NO  | YES |

SQL>  
-- You should see two accepted baselines

24. Purge the plan baselines. Use the `purge_cc_baselines.sql` script.

```

SQL> @purge_cc_baseline.sql
SQL> set echo on
SQL>
SQL> variable cnt number;
SQL>
SQL> exec :cnt := dbms_spm.drop_sql_plan_baseline(-
> '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
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.

```
-- Execute the below Statement

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>
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.

```
-- Execute the below Statement

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. Then verify that the 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;
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 SQL_0a9d872600ece455 |        |     |     |     |     |

```

select /*SPM_USE*/ * from sh.sales
where quantity_sold > 40 order by prod_id
SQL_PLAN_0p7c74s0ftt2p11df68d0 MANUAL-LOAD YES YES NO YES
7.6492E+17 SQL_0a9d872600ece455
select /*SPM_USE*/ * from sh.sales
where quantity_sold > 40 order by prod_id
SQL_PLAN_0p7c74s0ftt2p54bc8843 MANUAL-LOAD YES YES NO YES

SQL>
SQL>
```

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 |

Note

- SQL plan baseline "SQL_PLAN_0p7c74s0ftt2p54bc8843" used for
this statement

14 rows selected.

SQL>
SQL>
```

30. Disable this plan baseline, and check whether the system uses the other plan baseline when executing the statement again. Use the `check_baseline_used.sql` script. What do you observe?

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

SQL_0a9d872600ece455 SQL_PLAN_0p7c74s0ftt2p11df68d0
SQL_0a9d872600ece455 SQL_PLAN_0p7c74s0ftt2p54bc8843

SQL>
SQL>
SQL> exec :cnt := dbms_spm.alter_sql_plan_baseline(-
 > sql_handle => 'SQL_0a9d872600ece455', -
 > plan_name => 'SQL_PLAN_0p7c74s0ftt2p54bc8843', -
 > 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 SQL_0a9d872600ece455
select /*SPM_USE*/ * from sh.sales
where quantity_sold > 40 order by prod_id
SQL_PLAN_0p7c74s0ftt2p11df68d0 MANUAL-LOAD YES YES NO YES

7.6492E+17 SQL_0a9d872600ece455
select /*SPM_USE*/ * from sh.sales
where quantity_sold > 40 order by prod_id
SQL_PLAN_0p7c74s0ftt2p54bc8843 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
-----+

```

Note

-----  
- SQL plan baseline "SQL\_PLAN\_0p7c74s0ftt2p11df68d0" used for  
this statement

16 rows selected.

31. Disable the 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>
SQL> variable cnt number;
SQL>
SQL> exec :cnt := dbms_spm.alter_sql_plan_baseline(-
> sql_handle => 'SQL_0a9d872600ece455', -
> plan_name =>
'SQL_PLAN_0p7c74s0ftt2p11df68d0', -
> 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 SQL_0a9d872600ece455
select /*SPM_USE*/ * from sh.sales
where quantity_sold > 40 order by prod_id
SQL_PLAN_0p7c74s0ftt2p11df68d0 MANUAL-LOAD NO YES NO YES

7.6492E+17 SQL_0a9d872600ece455
select /*SPM_USE*/ * from sh.sales
where quantity_sold > 40 order by prod_id
SQL_PLAN_0p7c74s0ftt2p54bc8843 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 |

| 0 | SELECT STATEMENT |
| 1 | SORT ORDER BY |
| 2 | PARTITION RANGE ALL |
| 3 | TABLE ACCESS FULL | SALES |

10 rows selected.
```

SQL&gt;

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(
> '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
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. 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 nested loops 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 |

| 0 | SELECT STATEMENT | | | | 13
893 (1) | 00:00:1
1 | | |
| 1 | HASH GROUP BY | | | | 13
893 (1) | 00:00:1
1 | | |
```

```

| 2 | NESTED LOOPS | | 1 | 13 |
892 (1) | 00:00:1
1 | | |
| * 3 | TABLE ACCESS FULL | CUSTOMERS | 1 | 5 |
405 (1) | 00:00:0
5 | | |
| 4 | PARTITION RANGE ALL| | 1 | 8 |
488 (2) | 00:00:0
6 | 1 | 28 |
| * 5 | TABLE ACCESS FULL | SALES | 1 | 8 |
488 (2) | 00:00:0

PLAN_TABLE_OUTPUT

6 | 1 | 28 |


```

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>

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. 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;
```

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

```
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 SQL_18alef14c17f5b75
select /*+ USE_NL(s c) FULL(s) FULL(c) */
```

```
c.cust_id, sum(s.quantity_sold)
SQL_PLAN_1j8gg2m0ryqvpdc5f94e5 MANUAL-LOAD YES YES NO YES

SQL>
```

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 is not included here -
-- For a sample report see the sql_tuning_report.lst file -
```

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 SQL_18alef14c17f5b75
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
SQL_PLAN_1j8gg2m0ryqvpd5f94e5 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)               |          |              |
| 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>
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 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                            |                |     |     |     |     |
| SQL_PLAN_1j8gg2m0ryqvpa10c1dcf                | MANUAL-SQLTUNE | YES | YES | NO  | YES |
| 1.7750E+18 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                            |                |     |     |     |     |
| SQL_PLAN_1j8gg2m0ryqvpd5f94e5                 | 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 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;
```

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 |
| 6 | BITMAP INDEX RANGE SCAN | SALES_CUST_BIX |
| 7 | INDEX UNIQUE SCAN | CUSTOMERS_PK |

```

Note

-----

- SQL profile "SPM\_SQL\_PROF" used for this statement
- SQL plan baseline "SQL\_PLAN\_1j8gg2m0ryqvpa10c1dcf" used for this statement

19 rows selected.

### 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(
-> 'SQL_18alef14c17f5b75');

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
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>
SQL> alter system set optimizer_use_sql_plan_baselines=FALSE;

System altered.

SQL>
SQL> exit
```

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Error : You are not a Valid Partner use only

# **Workshop 1**

## **Chapter 14**

## Workshop 1: Overview

---

- Analyze a SQL query statement.
- Create a new index.
- Perform additional tasks.

### Scripts Used in the Workshop

- `setup01.sql`
- `ws01.sql`
- `w1_s1_e.sql`
- `ws01a.sql`
- `w1_s2_b.sql`
- `cleanup01.sql`

**Note:** Use the scripts provided for each workshop rather than copying and pasting the codes.

## Workshop 1: Enhancing the Performance of a SQL Query Statement

### Overview

In this workshop, you have to find a workaround to enhance performance. You analyze a poorly written SQL statement and perform additional tasks such as creating a function-based index, redesigning a simple table, and rewriting the SQL statement.

### Index Information

Index Name: HR.DEPT\_DEPT\_ID\_IDX1 :DEPT\_ID

### Tasks

1. Analyze a SQL query statement.

- a. Open Oracle SQL Developer.

On the desktop, double-click the SQL Developer desktop icon.



- b. Connect to hr schema.

The connection details are as follows:

Connection Name: hr\_connection

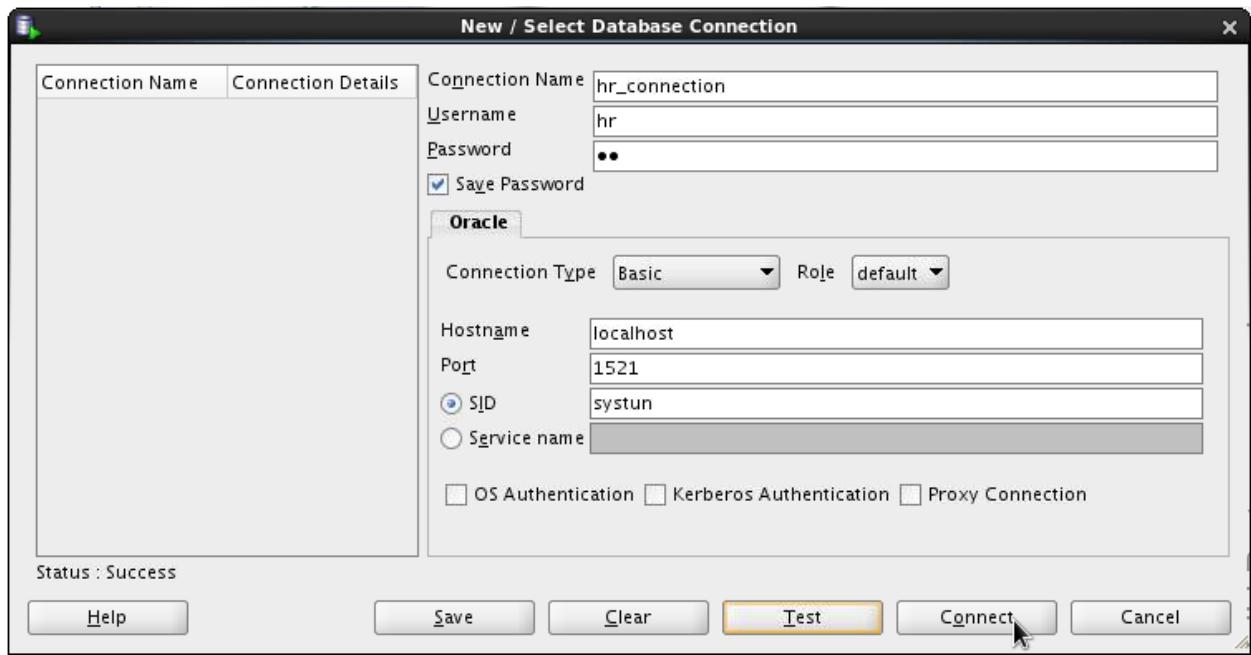
Username: hr

Password: hr

Hostname: localhost

Port : 1521

SID: systun



- c. Execute `setup01.sql` to set up the environment for this workshop.

**Note:** The scripts for this workshop are available in the workshops folder located in `/home/oracle/labs/workshops`.

`setup01.sql`

```
set echo on

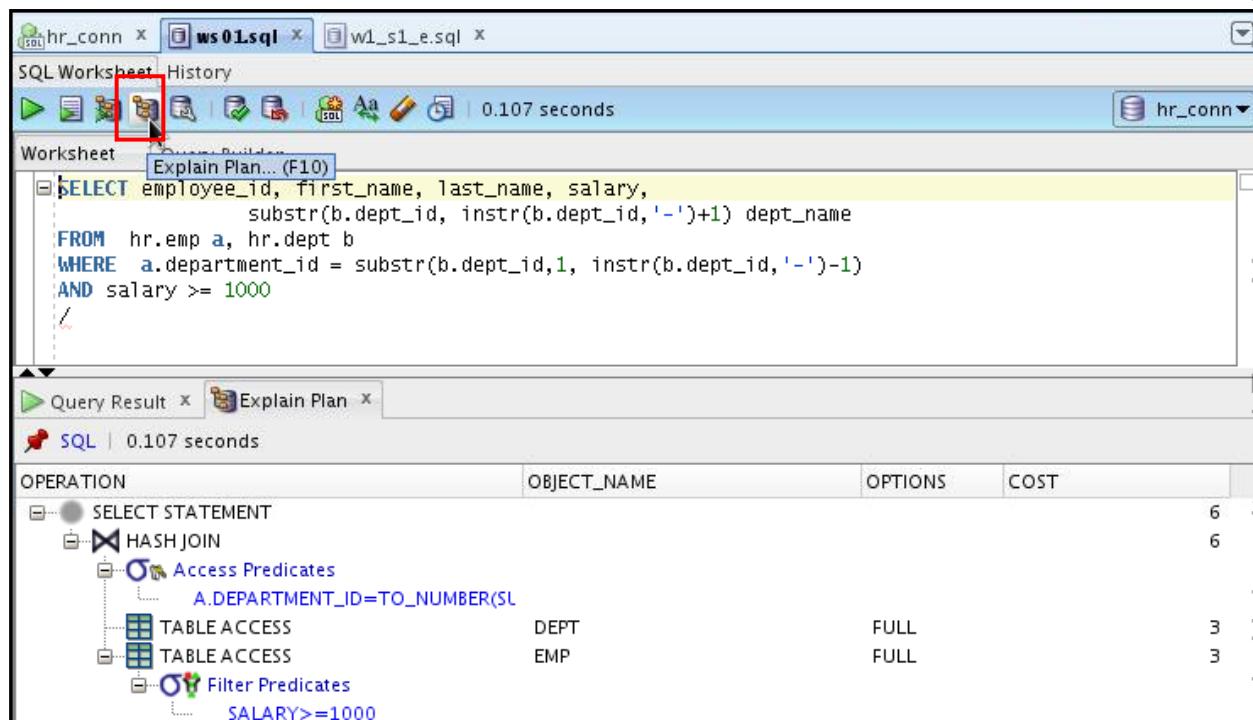
DROP TABLE hr.emp PURGE
/
CREATE TABLE hr.emp AS SELECT * FROM hr.employees
/
DESCRIBE HR.EMP
/
DROP TABLE hr.dept PURGE
/
CREATE TABLE hr.dept AS
SELECT (department_id||'-'||department_name) dept_id,
manager_id, location_id
FROM hr.departments
/
DROP INDEX hr.dept_dept_id_idx1
/
CREATE INDEX hr.dept_dept_id_idx1
ON hr.dept (dept_id)
NOLOGGING
COMPUTE STATISTICS
/
```

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

```
exec DBMS_STATS.GATHER_TABLE_STATS('HR','EMP')
/
exec DBMS_STATS.GATHER_TABLE_STATS('HR','DEPT')
/
Set echo off
```

- d. Review and check the execution plan of the SQL statement in ws01.sql. Click Explain Plan tool to check the plan.

```
SELECT employee_id, first_name, last_name, salary, substr(b.dept_id, instr(b.dept_id,'-')+1) dept_name
FROM hr.emp a, hr.dept b
WHERE a.department_id = substr(b.dept_id, 1, instr(b.dept_id,'-')-1)
AND salary >= 1000
```



**Note:** You can use either the EXPLAIN PLAN command, as shown in the script

w1\_s1\_e.sql, or the SQL Developer EXPLAIN PLAN tool icon in the SQL Worksheet to gather the execution plan.

- e. Gather the execution plan of the SQL statement by using the following EXPLAIN PLAN command. Use the w1\_s1\_e.sql file and click Run script icon.

```
EXPLAIN PLAN FOR SELECT employee_id, first_name, last_name,
salary, substr (b.dept_id, instr (b.dept_id,'-') +1) dept_name
FROM hr.emp a, hr.dept b
WHERE a.department_id = substr (b.dept_id, 1,
instr (b.dept_id,'-')-1)
AND salary >= 1000
/
SELECT * FROM TABLE (dbms_xplan.display);
```

The screenshot shows the Oracle SQL Developer interface. The top window is a SQL Worksheet titled 'w1\_s1.e.sql' with the query code. Below it is a 'Script Output' window showing the execution plan results.

**SQL Worksheet:**

```
EXPLAIN PLAN FOR SELECT employee_id, first_name, last_name, salary,
substr(b.dept_id, instr(b.dept_id,'-')+1) dept_name
FROM hr.emp a, hr.dept b
WHERE a.department_id = substr(b.dept_id,1, instr(b.dept_id,'-')-1)
AND salary >= 1000
/
Select * from table(dbms_xplan.display);
```

**Script Output:**

```
plan FOR succeeded.
PLAN_TABLE_OUTPUT
```

Plan hash value: 615168685

| Id | Operation         | Name | Rows | Bytes | Cost (%CPU) | Time     |
|----|-------------------|------|------|-------|-------------|----------|
| 0  | SELECT STATEMENT  |      | 263  | 11046 | 6 (0)       | 00:00:01 |
| 1  | HASH JOIN         |      | 263  | 11046 | 6 (0)       | 00:00:01 |
| 2  | TABLE ACCESS FULL | DEPT | 27   | 432   | 3 (0)       | 00:00:01 |
| 3  | TABLE ACCESS FULL | EMP  | 107  | 2782  | 3 (0)       | 00:00:01 |

Predicate Information (identified by operation id):

```
1 - access("A"."DEPARTMENT_ID"=TO_NUMBER(SUBSTR("B"."DEPT_ID",1,INSTR ("B"."DEPT_ID",'-')-1)))
3 - filter("SALARY">>=1000)
```

17 rows selected

**Note:** The EXPLAIN PLAN command is used to gather the execution plan.

- f. Review the execution plan. What do you observe?

- 1) Was the DEPT\_DEPT\_ID\_IDX1 index used?
- 2) If the index is not used, try to explain why the index is not used.

**Answer:**

- 1) No, the index DEPT\_DEPT\_ID\_IDX1 is not used.
  - 2) Internal transformation occurred due to the function in the indexed column.
2. Create a new index.

- a. Execute the following sample code to create a new index DEPT\_DEPT\_ID\_IDX2 that is usable by the optimizer or **you can use the ws01a.sql file**.

```
DROP INDEX hr.dept_dept_id_idx2
/
CREATE INDEX hr.dept_dept_id_idx2
ON hr.dept (substr (dept_id, 1, instr (dept_id, '-')-1))
NOLOGGING
COMPUTE STATISTICS
/
```

- b. Use the sample code to gather the execution plan of the SQL statement or **you can use the w1\_s2\_b.sql file**.

```
EXPLAIN PLAN FOR SELECT employee_id, first_name, last_name,
salary, substr (b.dept_id, instr (b.dept_id, '-') +1) dept_name
FROM hr.emp a, hr.dept b
WHERE a.department_id = substr (b.dept_id, 1,
instr (b.dept_id, '-')-1)
AND salary >= 1000
/
SELECT * FROM TABLE (dbms_xplan.display);
```

- c. Review the execution plan. What do you observe?
- 1) Was the DEPT\_DEPT\_ID\_IDX2 index used?
  - 2) If the index is not used, try to explain why the index is not used.

SQL Worksheet History

Worksheet Query Builder

```
EXPLAIN PLAN FOR SELECT employee_id, first_name, last_name, salary,
substr(b.dept_id, instr(b.dept_id, '-') + 1) dept_name
FROM hr.emp a, hr.dept b
WHERE a.department_id = substr(b.dept_id, 1, instr(b.dept_id, '-') - 1)
AND salary >= 1000
/
select * from table(dbms_xplan.display);
```

Script Output Task completed in 0.053 seconds

plan FOR succeeded.  
PLAN\_TABLE\_OUTPUT

Plan hash value: 615168685

| Id | Operation         | Name | Rows | Bytes | Cost (%CPU) | Time     |
|----|-------------------|------|------|-------|-------------|----------|
| 0  | SELECT STATEMENT  |      | 260  | 11960 | 6 (0)       | 00:00:01 |
| *  | HASH JOIN         |      | 260  | 11960 | 6 (0)       | 00:00:01 |
| 2  | TABLE ACCESS FULL | DEPT | 27   | 540   | 3 (0)       | 00:00:01 |
| *  | TABLE ACCESS FULL | EMP  | 107  | 2782  | 3 (0)       | 00:00:01 |

Predicate Information (identified by operation id):

```
1 - access("A"."DEPARTMENT_ID"=TO_NUMBER(SUBSTR("DEPT_ID",1,INSTR("DEPT_ID","-")-1)))
3 - filter("SALARY">>=1000)
```

17 rows selected

**Answer:**

- 1) No, the index DEPT\_DEPT\_ID\_IDX2 is not used.
  - 2) Data type mismatched. emp.department\_id (NUMBER)  
dept.dept\_id (CHAR).
3. Perform additional tasks.
- a. Perform additional tasks and rewrite the SQL statements to take advantage of an index.
    - Assume that you have a chance to redesign the DEPT table.
- Note:** To normalize the DEPT table, check all the columns with matching data type. You can create an index on the dept\_id column and check the execution plan.
- b. Gather the execution plan of the SQL statement by using the EXPLAIN PLAN command.

- c. Review the execution plan. What do you observe?
  - 1) Was an index used?
- d. Execute cleanup01.sql to clean up your environment for the next task.

```
DROP INDEX hr.dept_dept_id_idx1
/
DROP INDEX hr.dept_dept_id_idx2
/
DROP TABLE hr.emp PURGE
/
DROP TABLE hr.dept PURGE
/
```

- 4. After a function-based index is created, it is always chosen by the optimizer even if there is a condition that forces implicit data type conversion.
  - a. True
  - b. False

**Answer:** b

- 5. When computing selectivity on predicates of the form function (Column) = constant (for example, function-based index or built-in functions), the optimizer assumes a static selectivity value of 1 percent.
  - a. True
  - b. False

**Answer:** b

**Note:** The EXPLAIN PLAN command will be used in the subsequent practices.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Error : You are not a Valid Partner use only

## **Workshop 2**

### **Chapter 15**

## Workshop 2: Overview

---

- Display an execution plan by using the EXPLAIN PLAN command.
- Display an execution plan by using the V\$ view.
- Analyze and fix the issue.

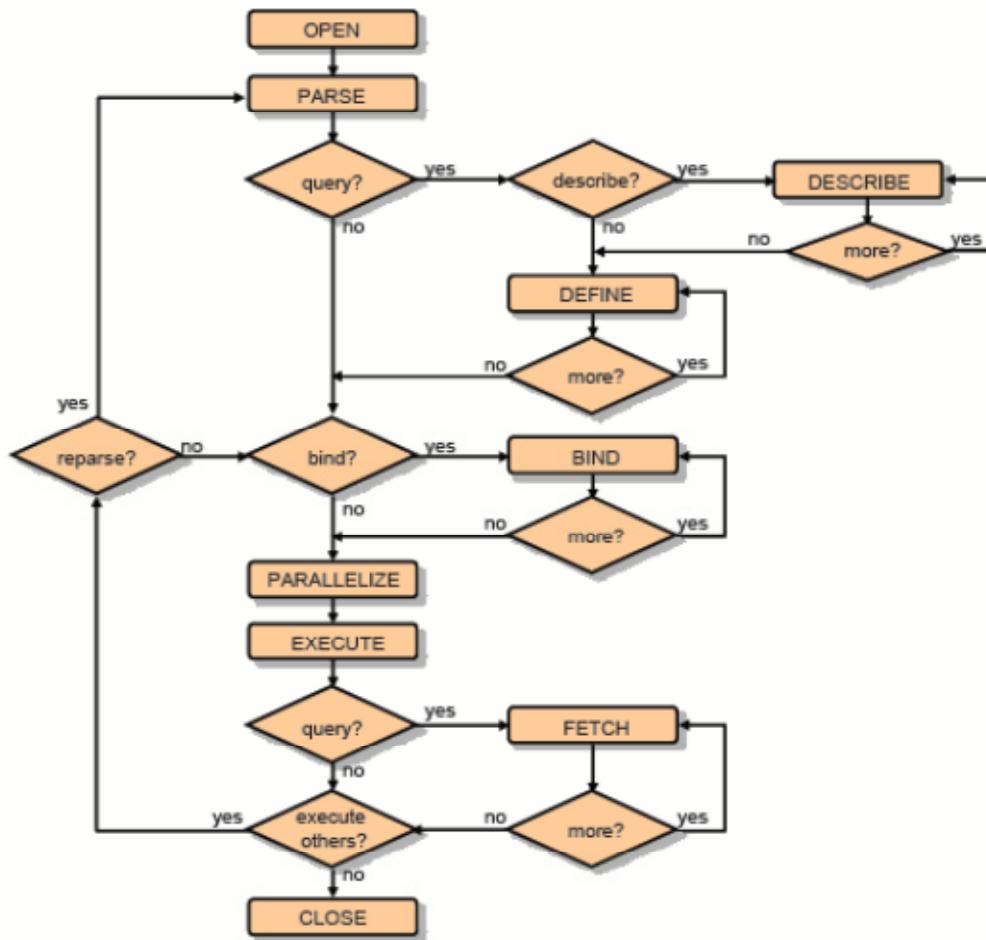
### Scripts Used in the Workshop

- `setup02.sql`
- `ws02.sql`
- `w2_s2_b.sql`
- `w2_s2_b1.sql`
- `cleanup02.sql`

## Workshop 2: Reviewing the Execution Steps of the SQL Statement

### Overview

This workshop is to review the execution steps of a SQL statement section. You review the SQL statement that uses a bind variable in the indexed column. You also analyze two execution plans of the SQL statement using the EXPLAIN PLAN command and the V\$ view. After completing this workshop, you should be able to understand why the two execution plans are different for the same SQL statement.



### Index Information

- Index Name: HR.EMP\_ID\_PK : EMP\_ID

### Tasks

- Display an execution plan using the EXPLAIN PLAN command.

- Connect to the hr schema in SQL Developer.
- Execute `setup02.sql` to set up the environment for this workshop.

**Note:** The scripts used in this workshop are available in the workshops folder located in `/home/oracle/labs/workshops`.

```
DROP TABLE hr.emp PURGE
/
CREATE TABLE emp (
 emp_id varchar2(5) CONSTRAINT emp_id_pk PRIMARY KEY,
 emp_fn varchar2(20),
 emp_ln varchar2(20),
 dept_id varchar2(5)
)
/

INSERT INTO emp
SELECT employee_id, first_name, last_name, department_id from
employees
/
commit
/

exec DBMS_STATS.GATHER_TABLE_STATS ('HR','EMP', METHOD_OPT =>
'for all indexed columns', CASCADE => TRUE)
/
```

- Execute the `ws02.sql` script and analyze the execution plan by using the EXPLAIN PLAN command. Click the Run Script  icon to run the script.

- Was the `EMP_ID_IDX` index used?

```
variable empid number
exec :empid:= 190

EXPLAIN PLAN FOR
SELECT * FROM hr.emp WHERE emp_id = :empid
/
SELECT * FROM TABLE (dbms_xplan.display);
```

The screenshot shows an Oracle SQL Developer interface. In the top window (SQL Worksheet), an anonymous block is written:

```

variable empid number
exec :empid := 190

EXPLAIN PLAN FOR
SELECT * FROM hr.emp WHERE emp_id = :empid
/
Select * from table(dbms_xplan.display);

```

The bottom window (Script Output) shows the results of the execution:

```

anonymous block completed
plan FOR succeeded.
PLAN_TABLE_OUTPUT

Plan hash value: 1252232671

| Id | Operation | Name | Rows | Bytes | Cost (%CPU)| Time |
| 0 | SELECT STATEMENT | | | | 1 (0) | 00:00:01 |
| 1 | TABLE ACCESS BY INDEX ROWID | EMP | | | 1 (0) | 00:00:01 |
|* 2 | INDEX UNIQUE SCAN | EMP_ID_PK | | | 0 (0) | 00:00:01 |

Predicate Information (identified by operation id):

2 - access("EMP_ID"=:EMPID)

14 rows selected

```

**Answer:**

- 1) Yes, the EMP\_ID\_IDX index is used.
2. Display an execution plan by using the V\$ view.
  - a. Execute the SQL statement in the ws02.sql script again.
  - b. Use the following sample codes to review the execution plan by using the V\$SQLAREA view and format it by using the DBMS\_XPLAN package; **or you can use w2\_s2\_b.sql and w2\_s2\_b1.sql files.**

```

variable empid number
exec: empid: =190
SELECT * FROM hr.emp WHERE emp_id=:190
/
select sql_id, plan_hash_value, sql_text
from v$sqlarea
where sql_text = 'SELECT * FROM hr.emp WHERE emp_id=: empid';

```

```
select * from table(dbms_xplan.display_cursor(<<'Please Enter the
SQL_ID value'>>));
```

- 1) What is the SQL\_ID of the SQL statement?
- 2) What function of DBMS\_XPLAN is used to format and display the contents of execution plan of any loaded cursor?
- 3) Was the EMP\_ID\_IDX index used?

The screenshot shows the Oracle SQL Developer interface. In the top window (Worksheet), a script is being run:

```
variable empid number
exec :empid := 190

SELECT * FROM hr.emp WHERE emp_id = :empid
/

select sql_id, plan_hash_value,sql_text
from v$sqlarea
where sql_text='SELECT * FROM hr.emp WHERE emp_id = :empid';
```

In the bottom window (Script Output), the results are displayed:

```
anonymous block completed
EMP_ID EMP_FN EMP_LN DEPT_ID
----- -----
190 Timothy Gates 50

SQL_ID PLAN_HASH_VALUE SQL_TEXT
----- -----
Ovy06dgmbqh9q 3956160932 SELECT * FROM hr.emp WHERE emp_id = :empid
```

A red box highlights the SQL\_ID 'Ovy06dgmbqh9q' in the output.

The screenshot shows an Oracle SQL Worksheet interface. In the top pane, a query is entered:

```
SELECT * FROM TABLE(dbms_xplan.display_cursor('Ovy06dgmbqh9q'));
```

In the bottom pane, the results of the query are displayed. The title bar says "Script Output" and "Query R...". The status bar indicates "All Rows Fetched: 18 in 0.025 seconds". The results show the execution plan for the query:

```

PLAN_TABLE_OUTPUT
1 SQL_ID Ovy06dgmbqh9q, child number 0
2 -----
3 SELECT * FROM hr.emp WHERE emp_id = :empid
4
5 Plan hash value: 3956160932
6
7 -----
8 | Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time |
9 -----
10| 0 | SELECT STATEMENT | | | | | |
11| * 1 | TABLE ACCESS FULL| EMP | 1 | 21 | 3 (0) | 00:00:01 |
12 -----
13
14 Predicate Information (identified by operation id):
15 -----
16
17 1 - filter(TO_NUMBER("EMP_ID")=:EMPID)
18

```

### Answer:

- 1) Ovy06dgmbqh9q SQL\_ID
  - 2) dbms\_xplan.display\_cursor is used to format and display the contents of execution plan of any loaded cursor.
  - 3) No, EMP\_ID\_IDX index is not used.
3. Analyze and fix the issue.
- a. Discuss why the two plans shown in Task 1 and Task 2 are different.
- Note:** In Task 1, the plan is executed before peeking at the value in the bind variable. And in Task 2, plan is executed after peeking at the value in the bind variable. Data types are mismatched between the bind variable and indexed column.
- b. Determine a cause and fix it to use the emp\_id\_pk index.
  - c. Execute the cleanup02.sql script for the next workshop.

```
DROP TABLE hr.emp PURGE
/
```

4. What statements are true?

- a. The EXPLAIN PLAN command is used to generate an optimizer execution plan.
- b. The EXPLAIN PLAN command executes the statement.
- c. When a bind variable is used, the EXPLAIN PLAN command peeks at the value in the bind variable.

**Answer:** a

5. The following steps are involved in query execution. Fill in the blank.

Parse → ( \_\_\_\_\_ ) → Execute → Fetch

**Answer:** Bind

# **Workshop 3**

## **Chapter 16**

## Workshop 3

---

### Workshop Overview

In the workshop, you will perform the following tasks:

- Perform an initial setup.
- Examine the indexed ORDER BY columns.

### Scripts Used in the Workshop

- `setup03.sql`
- `ws03.sql`
- `w3_s1_c.sql`
- `w3_s2_a.sql`
- `w3_s2_b.sql`
- `w3_s2_b1.sql`
- `w3_s2_c.sql`
- `w3_s2_d.sql`
- `w3_s3_a.sql`
- `w3_s3_b.sql`
- `w3_s3_b1.sql`
- `cleanup03.sql`

## Workshop 3: Learn to Tune Sort Operation Using an Index in the ORDER BY Clauses

---

### Overview

This workshop shows the possible usage of an index in the ORDER BY clause to tune a sort operation. You perform several tasks to be able to use the indexed ORDER BY column. After all tasks are done, you verify if the index always produces a better plan cost.

| Table Name | Column Name         | Constraint  |
|------------|---------------------|-------------|
| SH.CUST    | CUST_ID             | PRIMARY KEY |
|            | CUST_FN             |             |
|            | CUST_LN             |             |
|            | CUST_CREDIT_LIMIT   |             |
|            | CUST_STATE_PROVINCE |             |

### Tasks

1. Initial Setup
  - a. Connect to the sh schema in SQL Developer.  
Connection details are as follows:  
Connection Name: sh\_connection  
Username: sh  
Password: sh  
Hostname: localhost  
SID: systun
  - b. Execute the `setup03.sql` to set up the environment for this workshop.

```
DROP TABLE sh.cust PURGE
/
CREATE TABLE sh.cust (
 cust_id number primary key,
 cust_first_name varchar(20),
 cust_last_name varchar(40),
 cust_credit_limit number,
 cust_state_province varchar2 (40)
)
/
INSERT INTO sh.cust
SELECT cust_id, cust_first_name, cust_last_name,
cust_credit_limit, cust_state_province
FROM sh.customers
```

```
/

commit
/

CREATE INDEX sh.cust_cust_state_province_idx
ON sh.cust (cust_state_province)
NOLOGGING
COMPUTE STATISTICS
/

exec dbms_stats.gather_table_stats('SH','CUST',method_opt=>'for
all indexed columns', cascade=>true)
/

DESC sh.cust
/
```

**Note:** The scripts used in this workshop are available under workshops folder located in /home/oracle/labs/workshops.

Make sure sh user has the following privilege:

```
GRANT DBA to sh;
```

If not, connect as **sys** user in SQL Developer and grant the privilege to sh.

- c. Review the ws03.sql file. Gather the execution plan of the SQL statement using the EXPLAIN PLAN command given below or **you can use w3\_s1\_c.sql file.**

```
EXPLAIN PLAN FOR SELECT cust_first_name, cust_last_name,
cust_credit_limit
FROM sh.cust
WHERE cust_state_province = 'NJ'
ORDER BY cust_last_name, cust_first_name
/
SELECT * FROM TABLE (dbms_xplan.display);
```

- 1) What is the name of the index used?
- 2) What is the total cost?

The screenshot shows an Oracle SQL Worksheet interface. In the top tab bar, there are three tabs: 'sh\_conn' (selected), 'ws03.sql', and 'w3\_s1\_c.sql'. The main area is titled 'Worksheet' and contains the following SQL code:

```

EXPLAIN PLAN FOR SELECT cust_first_name, cust_last_name, cust_credit_limit
FROM sh.cust
WHERE cust_state_province = 'NJ'
ORDER BY cust_last_name, cust_first_name
/
SELECT * FROM TABLE(dbms_xplan.display);

```

Below the code, the 'Script Output' pane displays the execution results:

```

plan FOR succeeded.
PLAN_TABLE_OUTPUT

Plan hash value: 3273013652

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time |
|---|---|---|---|---|---|---|
| 0 | SELECT STATEMENT | | 209 | 6897 | 78 (0)| 00:00:01 |
| 1 | SORT ORDER BY | | 209 | 6897 | 78 (0)| 00:00:01 |
| 2 | TABLE ACCESS BY INDEX ROWID BATCHED | CUST | 209 | 6897 | 78 (0)| 00:00:01 |
|* 3 | INDEX RANGE SCAN | CUST_CUST_STATE_PROVINCE_IDX | 209 | 1 | 1 (0)| 00:00:01 |

Predicate Information (identified by operation id):

3 - access("CUST_STATE_PROVINCE"='NJ')

15 rows selected

```

### Answer:

- 1) The CUST\_CUST\_STATE\_PROVINCE\_IDX index is used.
- 2) Cost is 78.

**Note:** Your cost might differ depending on your environment.

2. Examine the indexed ORDER BY columns.
  - a. Use the following sample code to create a composite index, CUST\_FN\_LN\_IDX or you can use w3\_s2\_a.sql file.

**Note:** Index order: cust\_first\_name, cust\_last\_name

```

DROP INDEX sh.cust_fn_ln_idx
/
CREATE INDEX sh.cust_fn_ln_idx
ON sh.cust (cust_first_name, cust_last_name)
/

```

The screenshot shows the Oracle SQL Developer interface. In the top toolbar, there are icons for running, saving, and zooming, followed by the text "0.131 seconds". Below the toolbar, the tabs "Worksheet" and "Query Builder" are visible, with "Worksheet" being active. The main area contains the following SQL code:

```
DROP INDEX sh.cust_fn_ln_idx
/
CREATE INDEX sh.cust_fn_ln_idx
ON sh.cust (cust_first_name, cust_last_name)
/
```

Below the code, a "Script Output" window is open. It shows the execution time "Task completed in 0.131 seconds" and an error message:

```
Error starting at line 1 in command:
DROP INDEX sh.cust_fn_ln_idx
Error report:
SQL Error: ORA-01418: specified index does not exist
01418. 00000 - "specified index does not exist"
*Cause:
*Action:
index SH.CUST_FN_LN_IDX created.
```

- b. Gather and analyze the execution plan of the SQL statement again using the EXPLAIN PLAN command or you can use w3\_s2\_b.sql file.

```
EXPLAIN PLAN FOR SELECT cust_first_name, cust_last_name,
cust_credit_limit
FROM sh.cust
WHERE cust_state_province = 'NJ'
ORDER BY cust_last_name, cust_first_name
/

SELECT * FROM TABLE (dbms_xplan.display);
```

```

EXPLAIN PLAN FOR SELECT cust_first_name, cust_last_name, cust_credit_limit
FROM sh.cust
WHERE cust_state_province = 'NJ'
ORDER BY cust_last_name, cust_first_name
/
SELECT * FROM TABLE(dbms_xplan.display);

```

Script Output

Task completed in 0.031 seconds

plan FOR succeeded.

PLAN\_TABLE\_OUTPUT

Plan hash value: 3273013652

| Id   Operation                          | Name                         | Rows | Bytes | Cost (%CPU) | Time     |
|-----------------------------------------|------------------------------|------|-------|-------------|----------|
| 0   SELECT STATEMENT                    |                              | 209  | 6897  | 78 (0)      | 00:00:01 |
| 1   SORT ORDER BY                       |                              | 209  | 6897  | 78 (0)      | 00:00:01 |
| 2   TABLE ACCESS BY INDEX ROWID BATCHED | CUST                         | 209  | 6897  | 78 (0)      | 00:00:01 |
| /* 3   INDEX RANGE SCAN                 | CUST_CUST_STATE_PROVINCE_IDX | 209  | 1     | (0)         | 00:00:01 |

Predicate Information (identified by operation id):

3 - access("CUST\_STATE\_PROVINCE">'NJ')

15 rows selected

- Was the CUST\_FN\_LN\_IDX index used? If not, add an index hint to allow the optimizer to use the cust\_fn\_ln\_idx index and gather the execution plan using the EXPLAIN PLAN command. You can use w3\_s2\_b1.sql.

```

EXPLAIN PLAN FOR SELECT /*+ index (cust cust_fn_ln_idx) */
cust_first_name, cust_last_name, cust_credit_limit
FROM sh.cust
WHERE cust_state_province = 'NJ'
ORDER BY cust_last_name, cust_first_name
/
SELECT * FROM TABLE (dbms_xplan.display);

```

The screenshot shows the Oracle SQL Developer interface. In the top window (SQL Worksheet), the following EXPLAIN PLAN query is run:

```
EXPLAIN PLAN FOR SELECT /*+ index (cust cust_fn_ln_idx) */ cust_first_name, cust_last_name, cust_credit_limit
FROM sh.cust
WHERE cust_state_province = 'NJ'
ORDER BY cust_last_name, cust_first_name
/
SELECT * FROM TABLE(dbms_xplan.display);
```

The output window (Script Output) displays the execution results:

```
plan FOR succeeded.
PLAN_TABLE_OUTPUT

Plan hash value: 3273013652

-----| Id | Operation | Name | Rows | Bytes | Cost (%CPU)| Time |
-----| 0 | SELECT STATEMENT | | 209 | 6897 | 78 (0) | 00:00:01 |
| 1 | SORT ORDER BY | | 209 | 6897 | 78 (0) | 00:00:01 |
| 2 | TABLE ACCESS BY INDEX ROWID BATCHED | CUST | 209 | 6897 | 78 (0) | 00:00:01 |
|* 3 | INDEX RANGE SCAN | CUST_CUST_STATE_PROVINCE_IDX | 209 | 1 | 1 (0) | 00:00:01 |

Predicate Information (identified by operation id):

 3 - access("CUST_STATE_PROVINCE"='NJ')

15 rows selected
```

- 2) Was a sort operation avoided?
- 3) Was the cost of the plan better than the one in Task 1?

**Note:** sh.cust\_fn\_ln\_idx is not used. At least, one of the indexed columns in the ORDER BY clause should have the NOT NULL constraint.

- c. If the index is not used, take additional tasks to use the indexed ORDER BY columns.

**Hint:** Consider adding the NOT NULL constraint.

Use the sample code or you can use the w3\_s2\_c.sql file.

```
ALTER TABLE sh.cust MODIFY (cust_last_name NOT NULL)
/
```

- d. After step c is performed, repeat Step b to verify if the composite index is used. Use the w3\_s2\_d.sql file.

**Note:** If the composite index is not used even after adding a NOT NULL constraint to the ORDER BY columns, then move to Task 3.

```

EXPLAIN PLAN FOR SELECT cust_first_name, cust_last_name, cust_credit_limit
FROM sh.cust
WHERE cust_state_province = 'NJ'
ORDER BY cust_last_name, cust_first_name
/
SELECT * FROM TABLE(dbms_xplan.display);

plan FOR succeeded.
PLAN_TABLE_OUTPUT

Plan hash value: 3273013652

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time |
| 0 | SELECT STATEMENT | | 209 | 6897 | 78 (0) | 00:00:01 |
| 1 | SORT ORDER BY | | 209 | 6897 | 78 (0) | 00:00:01 |
| 2 | TABLE ACCESS BY INDEX ROWID BATCHED | CUST | 209 | 6897 | 78 (0) | 00:00:01 |
|* 3 | INDEX RANGE SCAN | CUST_CUST_STATE_PROVINCE_IDX | 209 | 1 (0) | 00:00:01 |

Predicate Information (identified by operation id):

3 - access("CUST_STATE_PROVINCE"='NJ')

15 rows selected

```

3. Examine the indexed ORDER BY columns.

- a. Use the sample code to create another composite index, CUST\_LN\_FN\_IDX, on the ORDER BY columns or you can use the w3\_s3\_a.sql file.

**Note:** Index order: cust\_last\_name, cust\_first\_name

```

DROP INDEX sh.cust_ln_fn_idx
/
CREATE INDEX sh.cust_ln_fn_idx
ON sh.cust (cust_last_name, cust_first_name)
/

```

The screenshot shows an Oracle SQL Worksheet interface. In the top pane, there is a code editor with the following SQL script:

```
SQL Worksheet History
Worksheet Query Builder
DROP INDEX sh.cust_ln_fn_idx
/
CREATE INDEX sh.cust_ln_fn_idx
ON sh.cust (cust_last_name, cust_first_name)
/
```

In the bottom pane, the "Script Output" window displays the results of the execution:

```
Script Output x
Task completed in 0.171 seconds
Error starting at line 1 in command:
DROP INDEX sh.cust_ln_fn_idx
Error report:
SQL Error: ORA-01418: specified index does not exist
01418. 00000 - "specified index does not exist"
*Cause:
*Action:
index SH.CUST_LN_FN_IDX created.
```

- b. Gather and analyze the execution plan of the SQL statement again using the EXPLAIN PLAN command. You can use the w3\_s3\_b.sql file.

```
EXPLAIN PLAN FOR SELECT cust_first_name, cust_last_name,
cust_credit_limit
FROM sh.cust
WHERE cust_state_province = 'NJ'
ORDER BY cust_last_name, cust_first_name
/
SELECT * FROM TABLE (dbms_xplan.display);
```

The screenshot shows the Oracle SQL Worksheet interface. In the top tab bar, there are three tabs: 'sh\_conn' (active), 'w3\_s2\_c.sql', and 'w3\_s3\_b.sql'. The main area is titled 'Worksheet' and contains a 'Query Builder' section with the following SQL code:

```
EXPLAIN PLAN FOR SELECT cust_first_name, cust_last_name, cust_credit_limit
FROM sh.cust
WHERE cust_state_province='NJ'
ORDER BY cust_last_name, cust_first_name
/
SELECT * FROM TABLE(dbms_xplan.display);
```

Below the code, the 'Script Output' tab is active, showing the results of the EXPLAIN PLAN command:

```
plan FOR succeeded.
PLAN_TABLE_OUTPUT

Plan hash value: 3273013652

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time |
| 0 | SELECT STATEMENT | | 209 | 6897 | 78 (0) | 00:00:01 |
| 1 | SORT ORDER BY | | 209 | 6897 | 78 (0) | 00:00:01 |
| 2 | TABLE ACCESS BY INDEX ROWID BATCHED | CUST | 209 | 6897 | 78 (0) | 00:00:01 |
|* 3 | INDEX RANGE SCAN | CUST_CUST_STATE_PROVINCE_IDX | 209 | | 1 (0) | 00:00:01 |

Predicate Information (identified by operation id):

3 - access("CUST_STATE_PROVINCE"='NJ')

15 rows selected
```

- 1) Was the CUST\_LN\_FN\_IDX index used? If not, add an index hint to allow the optimizer to use the cust\_ln\_fn\_idx index and gather the execution plan using the EXPLAIN PLAN command. You can use the w3\_s3\_b1.sql file.

```
EXPLAIN PLAN FOR SELECT /*+ index (cust cust_ln_fn_idx) */
cust_first_name, cust_last_name, cust_credit_limit
FROM sh.cust
WHERE cust_state_province = 'NJ'
ORDER BY cust_last_name, cust_first_name
/
SELECT * FROM TABLE (dbms_xplan.display);
```

```

EXPLAIN PLAN FOR SELECT /*+ index (cust cust_ln_fn_idx) */ cust_first_name, cust_last_name, cust_credit_limit
FROM sh.cust
WHERE cust_state_province = 'NJ'
ORDER BY cust_last_name, cust_first_name
/
SELECT * FROM TABLE(dbms_xplan.display);

```

Script Output x  
Task completed in 0.032 seconds

plan FOR succeeded.  
PLAN\_TABLE\_OUTPUT

Plan hash value: 2664602889

| Id | Operation                   | Name           | Rows  | Bytes | Cost (%CPU) | Time     |
|----|-----------------------------|----------------|-------|-------|-------------|----------|
| 0  | SELECT STATEMENT            |                | 209   | 6897  | 9433 (1)    | 00:00:01 |
| 1  | TABLE ACCESS BY INDEX ROWID | CUST           | 209   | 6897  | 9433 (1)    | 00:00:01 |
| 2  | INDEX FULL SCAN             | CUST_LN_FN_IDX | 55500 |       | 194 (0)     | 00:00:01 |

Predicate Information (identified by operation id):

1 - filter("CUST\_STATE\_PROVINCE"='NJ')

14 rows selected

- 2) Was a sort operation avoided?
- 3) Was the cost of the plan better than the one in Task 1?

**Note:** The CUST\_LN\_FN\_IDX index is used. However, you learned that using the indexed ORDER by column that avoids a sort operation doesn't always produce better performance.

#### 4. Your Observation

**Note:** You have performed several tasks to see the possible usage of an index in the ORDER BY clause to tune a sort operation. Take a minute to answer the questions below based on your observation.

- a. What do you observe?
- b. Execute the cleanup03.sql script for the next workshop.

```

DROP INDEX sh.cust_cust_state_province_idx
/
DROP INDEX sh.cust_fn_ln_idx
/
DROP INDEX sh.cust_ln_fn_idx
/
DROP TABLE sh.cust PURGE
/

```

5. When using the indexed ORDER BY column to avoid a sort operation, you should consider the following:
  - a. All the order by column should be in the composite index.
  - b. Order by columns should be of the same order as that of the composite index.
  - c. At least one of the composite columns should have the NOT NULL constraint.
  - d. All composite columns should have the NOT NULL constraint.
  - e. Using the indexed ORDER BY column that avoids a sort operation always produces better performance.

**Answer:** a, b, and c

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Error : You are not a Valid Partner use only

## **Workshop 4**

### **Chapter 17**

## Workshop 4: Overview

---

### Workshop Overview

In this workshop, you will perform the following tasks:

- Identify a poorly written SQL statement.
- Tune the SQL statement by rewriting it.

### Scripts Used in the Workshop

- `setup04.sql`
- `ws04.sql`
- `w4_s1_d.sql`
- `w4_s2_a.sql`
- `w4_s2_b.sql`

## Workshop 4: Identifying and Tuning a Poorly Written SQL Statement

### Overview

In this workshop, you assume that you have identified a poorly running SQL statement. You already noticed that the root cause of the issue is the table design. However, re-creating the table is not an option at this point. The table is already in use. Try to tune the SQL statement by rewriting it.

### Index Information

| Index Name | Columns        |
|------------|----------------|
| TIME_IDX   | YYYY + MM + DD |

### Tasks

1. Initial setup
  - a. Connect to the sh schema in SQL Developer.
  - b. Use setup04.sql to set up the environment for this workshop.

**Note:** The scripts used in this workshop are available in the workshops folder located in /home/oracle/labs/workshops.

```
DROP TABLE sh.sales1 PURGE
/
CREATE TABLE sh.sales1 (
 sales_id number primary key,
 prod_id number,
 YYYY char(4),
 mm char(2),
 dd char(2),
 amount_sold number(10,2)
)
/
DROP SEQUENCE sh.sales_id_seq
/
CREATE SEQUENCE sh.sales_id_seq
 START WITH 1
 INCREMENT BY 1
 NOCACHE
 NOCYCLE
/
INSERT INTO sh.sales1
```

```
SELECT sales_id_seq.nextval, prod_id,
 to_char (time_id, 'YYYY'), to_char (time_id, 'MM'), to_char
 (time_id, 'DD'), amount_sold
 FROM sh.sales
/
COMMIT
/
DROP TABLE sh.prod1 PURGE
/
CREATE TABLE sh.prod1 AS SELECT * FROM sh.products
/
DROP INDEX sh.time_idx
/
CREATE INDEX sh.time_idx
 ON sh.sales1 (YYYY, MM, DD, sales_id)
 NOLOGGING
 COMPUTE STATISTICS
/
exec DBMS_STATS.GATHER_TABLE_STATS ('SH','SALES1', METHOD_OPT =>
'FOR ALL INDEXED COLUMNS', CASCADE => TRUE)
/
exec DBMS_STATS.GATHER_TABLE_STATS('SH','PROD1', METHOD_OPT =>
'FOR ALL INDEXED COLUMNS', CASCADE => TRUE)
/
DESC sh.sales1
DESC sh.prod1
```

- c. Review the SQL statement in ws04.sql.

```
SELECT (s.yyyy || s.mm || s.dd) TIME, s.prod_id, sum
(amount_sold)
 FROM sh.sales1 s, sh.prod1 p
 WHERE s.prod_id = p.prod_id
 AND s.yyyy || s.mm || s.dd BETWEEN '19980901' AND '19980907'
 AND p.prod_category = 'Hardware'
 GROUP BY s.prod_id, s.yyyy || s.mm || s.dd
 ORDER BY 1
```

/

- d. Gather the execution plan of the SQL statement by using the EXPLAIN PLAN command by using the following sample code or you can use the w4\_s1\_d.sql file.

```
EXPLAIN PLAN FOR SELECT (s.yyyy || s.mm || s.dd) TIME,
s.prod_id, sum (amount_sold)
FROM sh.sales1 s, sh.prod1 p
WHERE s.prod_id = p.prod_id
AND s.yyyy || s.mm || s.dd BETWEEN '19980901' AND '19980907'
AND p.prod_category = 'Hardware'
GROUP BY s.prod_id, s.yyyy || s.mm || s.dd
ORDER BY 1
/
SELECT * FROM TABLE (dbms_xplan.display);
```

- 1) Was the TIME\_IDX index used?
- 2) What is the total cost?

SQL Worksheet History

Worksheet Query Builder

```
EXPLAIN PLAN FOR SELECT (s.yyyy || s.mm || s.dd) TIME, s.prod_id, sum (amount_sold)
FROM sh.sales1 s, sh.prod1 p
WHERE s.prod_id = p.prod_id
AND s.yyyy || s.mm || s.dd BETWEEN '19980901' AND '19980907'
AND p.prod_category = 'Hardware'
GROUP BY s.prod_id, s.yyyy || s.mm || s.dd
ORDER BY 1
/
SELECT * FROM TABLE(dbms_xplan.display);
```

Script Output x

Task completed in 0.041 seconds

PLAN FOR succeeded.  
PLAN\_TABLE\_OUTPUT

Plan hash value: 281941987

| Id  | Operation         | Name   | Rows | Bytes | Cost (%CPU) | Time     |
|-----|-------------------|--------|------|-------|-------------|----------|
| 0   | SELECT STATEMENT  |        | 24   | 1056  | 1009 (2)    | 00:00:01 |
| 1   | SORT GROUP BY     |        | 24   | 1056  | 1009 (2)    | 00:00:01 |
| * 2 | HASH JOIN         |        | 459  | 20196 | 1009 (2)    | 00:00:01 |
| * 3 | TABLE ACCESS FULL | PROD1  | 14   | 294   | 3 (0)       | 00:00:01 |
| * 4 | TABLE ACCESS FULL | SALES1 | 2297 | 52831 | 1006 (2)    | 00:00:01 |

Predicate Information (identified by operation id):

```
2 - access("S"."PROD_ID"="P"."PROD_ID")
3 - filter("P"."PROD_CATEGORY"='Hardware')
4 - filter("S"."YYYY"||"S"."MM"||"S"."DD">>='19980901' AND
 "S"."YYYY"||"S"."MM"||"S"."DD"<='19980907')
```

19 rows selected

**Answer:**

- 1) The TIME\_IDX index is not used.
- 2) Cost is 1009 in this case.

**Note:** In this case, the cost is 1009. It might differ also.

2. Rewrite the SQL statement.

- a. If the TIME\_IDX is not used, rewrite and review the SQL statement in w4\_s2\_a.sql to take advantage of the index.

**Hint:** Modify the BETWEEN operator.

```
SELECT (s.yyyy || s.mm || s.dd) TIME, s.prod_id, sum
(amount_sold)
FROM sh.sales1 s, sh.prod1 p
WHERE s.prod_id = p.prod_id
AND (s.yyyy = '1998' AND s.mm = '09' AND s.dd BETWEEN '01' AND
'07')
AND p.prod_category = 'Hardware'
GROUP BY s.yyyy || s.mm || s.dd, s.prod_id
ORDER BY 1
/
```

- b. Use the sample code to gather the execution plan of the rewritten SQL statement by using the EXPLAIN PLAN command. You can use the w4\_s2\_b.sql file.

```
EXPLAIN PLAN FOR SELECT (s.yyyy || s.mm || s.dd) TIME,
s.prod_id, sum (amount_sold)
FROM sh.sales1 s, sh.prod1 p
WHERE s.prod_id = p.prod_id
AND (s.yyyy = '1998' AND s.mm = '09' AND s.dd BETWEEN '01' AND
'07')
AND p.prod_category = 'Hardware'
GROUP BY s.yyyy || s.mm || s.dd, s.prod_id
ORDER BY 1
/
SELECT * FROM TABLE (dbms_xplan.display);
```

- 1) Was the TIME\_IDX index used?
- 2) What is the total cost?

The screenshot shows the Oracle SQL Worksheet interface. In the top tab bar, there are two tabs: 'w4\_s2\_a.sql' and 'w4\_s2\_b.sql'. The 'w4\_s2\_b.sql' tab is active. Below the tabs, the title bar includes 'SQL Worksheet History' and a toolbar with various icons. The main area is divided into two panes. The left pane contains the SQL code:

```

EXPLAIN PLAN FOR SELECT (s.yyyy || s.mm || s.dd) TIME, s.prod_id, sum(amount_sold)
FROM sh.sales1 s, sh.prod1 p
WHERE s.prod_id = p.prod_id
AND (s.yyyy = '1998' AND s.mm = '09' AND s.dd BETWEEN '01' AND '07')
AND p.prod_category = 'Hardware'
GROUP BY s.yyyy || s.mm || s.dd, s.prod_id
ORDER BY 1
/
SELECT * FROM TABLE(dbms_xplan.display);

```

The right pane displays the execution results. It starts with a 'Script Output' section showing the message 'Task completed in 0.043 seconds'. Below that, it shows the plan for the succeeded query, including the plan hash value (41534641). A detailed execution plan table follows:

| Id   Operation                             | Name     | Rows                                | Bytes | Cost | (%CPU) | Time |
|--------------------------------------------|----------|-------------------------------------|-------|------|--------|------|
| 0   SELECT STATEMENT                       |          | 2   88   175   (0)   00:00:01       |       |      |        |      |
| 1   SORT GROUP BY                          |          | 2   88   175   (0)   00:00:01       |       |      |        |      |
| /* 2   HASH JOIN                           |          | 548   24112   175   (0)   00:00:01  |       |      |        |      |
| /* 3   TABLE ACCESS FULL                   | PROD1    | 14   294   3   (0)   00:00:01       |       |      |        |      |
| /* 4   TABLE ACCESS BY INDEX ROWID BATCHED | SALES1   | 2739   62997   172   (0)   00:00:01 |       |      |        |      |
| /* 5   INDEX RANGE SCAN                    | TIME_IDX | 2739     13   (0)   00:00:01        |       |      |        |      |

Below the table, the 'Predicate Information (identified by operation id):' section lists three predicates:

- 2 - access("S"."PROD\_ID"="P"."PROD\_ID")
- 3 - filter("P"."PROD\_CATEGORY"='Hardware')
- 5 - access("S"."YYYY"='1998' AND "S"."MM"='09' AND "S"."DD">>='01' AND "S"."DD" <='07')

At the bottom of the results pane, it says '19 rows selected'.

### Answer:

- 1) The TIME\_IDX index is used.

**Note:** Re-designing the SALES1 table would be a better choice. But if the table is already in use, you could rewrite the SQL statement to take advantage of the index.

- 2) The total cost is 175.

**Note:** In this case, the cost is 175. It might differ also.

### 3. Your observation

What was your observation?

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Error : You are not a Valid Partner use only

# **Workshop 5**

## **Chapter 18**

## Workshop 5: Overview

---

### Workshop Overview

In this workshop, you will perform the following tasks:

- You write several SQL statements that return the same output to see which SQL statement is more efficient.
- Examine the effects of changing the column order in a composite index.

### Scripts Used in the Workshop

- `setup05.sql`
- `ws05.sql`
- `w5_s1_c.sql`
- `w5_s2_b.sql`
- `w5_s3_b.sql`
- `w5_s4_a.sql`
- `w5_s4_b.sql`
- `w5_s5_a.sql`
- `w5_s5_c.sql`
- `w5_s5_d.sql`
- `cleanup05.sql`

## Workshop 5: Effects of Changing the Column Order in a Composite Index

### Overview

In this workshop, you write several SQL statements that return the same output to see which SQL statement is more efficient. You also examine the effects of changing the column order in a composite index.

### Index Information

| Index Name            | Columns                                 | Initial Status |
|-----------------------|-----------------------------------------|----------------|
| cust_state_idx        | cust_state_province                     | Visible        |
| cust_income_state_idx | cust_income_level + cust_state_province | Invisible      |
| cust_state_gender_idx | cust_state_province + cust_gender       | Invisible      |

### Tasks

1. Setup
  - a. Connect to the sh schema in SQL Developer.
  - b. Use the `setup05.sql` script to set up your environment for this task.

**Note:** The scripts used in this workshop are available under workshops folder located in  
`/home/oracle/labs/workshops`.

```
DROP TABLE sh.cust PURGE
/
CREATE TABLE sh.cust AS SELECT * FROM sh.customers
/

DROP INDEX sh.cust_state_idx
/

CREATE INDEX sh.cust_state_idx
ON sh.cust (cust_state_province)
NOLOGGING
COMPUTE STATISTICS
/

DROP INDEX sh.cust_state_gender_idx
/

CREATE INDEX sh.cust_state_gender_idx
ON sh.cust (cust_state_province, cust_gender)
NOLOGGING
```

```

COMPUTE STATISTICS
/
ALTER INDEX sh.cust_state_gender_idx INVISIBLE
/
DROP INDEX sh.cust_income_state_idx
/
CREATE INDEX sh.cust_income_state_idx
ON sh.cust(cust_income_level,cust_state_province)
NOLOGGING
COMPUTE STATISTICS
/
ALTER INDEX sh.cust_income_state_idx INVISIBLE
/
exec DBMS_STATS.GATHER_TABLE_STATS('SH','CUST', METHOD_OPT =>
'FOR ALL INDEXED COLUMNS', CASCADE => TRUE)
/

```

- c. Make two indexes CUST\_INCOME\_STATE\_IDX and CUST\_STATE\_GENDER\_IDX visible to the optimizer by using the following code or you can use the w5\_s1\_c.sql file.

```

ALTER INDEX sh.cust_income_state_idx VISIBLE
/
ALTER INDEX sh.cust_state_gender_idx VISIBLE
/

```

2. Test the first query.

- a. Review the SQL statement (Query 1) in ws05.sql. In the first query, you transformed the indexed columns, CUST\_STATE\_PROVINCE and CUST\_GENDER explicitly on purpose.

```

SELECT cust_first_name, cust_last_name, cust_income_level,
cust_credit_limit
FROM sh.cust
WHERE cust_income_level like 'F%'
AND cust_state_province||'' in ('NJ', 'NY')
AND cust_gender ||'' = 'M'

```

- b. Execute the following sample code and gather the execution plan of the SQL statement by using the EXPLAIN PLAN command. You can use the w5\_s2\_b.sql file.

```

EXPLAIN PLAN FOR SELECT cust_first_name, cust_last_name,
cust_income_level, cust_credit_limit
FROM sh.cust
WHERE cust_income_level like 'F%'
AND cust_state_province in ('NJ', 'NY')
AND cust_gender = 'M'
/
SELECT * FROM TABLE (dbms_xplan.display);

```

- 1) Which index was used?
- 2) What is the total cost?

The screenshot shows the Oracle SQL Worksheet interface. The top window is titled 'w5\_s2.b.sql' and contains the SQL code for the EXPLAIN PLAN query. Below it is a 'Script Output' window showing the execution results.

```

SQL Worksheet History
Worksheet Query Builder
sh_conn

EXPLAIN PLAN FOR SELECT cust_first_name, cust_last_name, cust_income_level, cust_credit_limit
FROM sh.cust
WHERE cust_income_level like 'F%'
AND cust_state_province||'' in ('NJ', 'NY')
AND cust_gender ||'' = 'M'
/
SELECT * FROM TABLE(dbms_xplan.display);

plan FOR succeeded.
PLAN_TABLE_OUTPUT
Plan hash value: 1367750724

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time |
| 0 | SELECT STATEMENT | | 106 | 5518 | 185 (0) | 00:00:01 |
|* 1 | TABLE ACCESS BY INDEX ROWID BATCHED | CUST | 106 | 5518 | 185 (0) | 00:00:01 |
|* 2 | INDEX RANGE SCAN | CUST_INCOME_STATE_IDX | 159 | 64 | 64 (0) | 00:00:01 |

Predicate Information (identified by operation id):
1 - filter("CUST_GENDER"||''='M')
2 - access("CUST_INCOME_LEVEL" LIKE 'F%')
 filter("CUST_INCOME_LEVEL" LIKE 'F%' AND ("CUST_STATE_PROVINCE"||''='NY' OR
 "CUST_STATE_PROVINCE"||''='NJ'))

```

17 rows selected

### Answer:

- 1) The CUST\_INCOME\_STATE\_IDX index is used.
- 2) The cost is 185.

**Note:** The cost is 185 in this case. It might differ also.

- c. Did the optimizer consider the CUST\_STATE\_GENDER\_IDX index? If not, try to explain why the index was not considered.

**Note:** The CUST\_STATE\_GENDER\_IDX index was not considered due to the transformed columns.

3. Test the second query.

- a. Rewrite the SQL statement (Query 2) to consider the CUST\_STATE\_GENDER\_IDX index as an option as well.

```
SELECT cust_first_name, cust_last_name, cust_income_level,
cust_credit_limit
FROM sh.cust
WHERE cust_income_level like 'F%'
AND cust_state_province in ('NJ', 'NY')
AND cust_gender = 'M'
AND rownum < 31
/
```

- b. Execute the following sample code and gather the execution plan of the SQL statement using the EXPLAIN PLAN command. You can use the w5\_s3\_b.sql file.

```
EXPLAIN PLAN FOR SELECT cust_first_name, cust_last_name,
cust_income_level, cust_credit_limit
FROM sh.cust
WHERE cust_income_level like 'F%'
AND cust_state_province in ('NJ', 'NY')
AND cust_gender = 'M'
AND rownum < 31
/
SELECT * FROM TABLE (dbms_xplan.display);
```

- 1) Which index was used?
- 2) What is the total cost?

The screenshot shows the Oracle SQL Developer interface. In the top window (Worksheet), the code is:

```

EXPLAIN PLAN FOR SELECT cust_first_name, cust_last_name, cust_income_level, cust_credit_limit
FROM sh.cust
WHERE cust_income_level like 'F%'
AND cust_state_province in ('NJ', 'NY')
AND cust_gender = 'M'
AND rownum < 31
/
SELECT * FROM TABLE(dbms_xplan.display);

```

In the bottom window (Script Output), the results are:

```

plan FOR succeeded.
PLAN_TABLE_OUTPUT

Plan hash value: 2872430508

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time |
| 0 | SELECT STATEMENT | | 30 | 1590 | 55 (0) | 00:00:01 |
|* 1 | COUNT STOPKEY | | | | | |
|* 2 | TABLE ACCESS BY INDEX ROWID BATCHED | CUST | 30 | 1590 | 55 (0) | 00:00:01 |
|* 3 | INDEX RANGE SCAN | CUST_INCOME_STATE_IDX | 159 | | 20 (0) | 00:00:01 |

Predicate Information (identified by operation id):
1 - filter(ROWNUM<31)
2 - filter("CUST_GENDER"='M')
3 - access("CUST_INCOME_LEVEL" LIKE 'F%')
 filter(("CUST_STATE_PROVINCE"='NJ' OR "CUST_STATE_PROVINCE"='NY') AND "CUST_INCOME_LEVEL" LIKE 'F%')

19 rows selected

```

### Answer:

- 1) The CUST\_INCOME\_STATE\_IDX index is used.
- 2) The cost is 55.

**Note:** The cost is 55 in this case. It might differ also.

- c. Did the optimizer consider the CUST\_STATE\_GENDER\_IDX index? If no, try to explain why the index was not considered in Task 3.

**Note:** In Task 3, the CUST\_STATE\_GENDER\_IDX index was considered, but was not an optimal access path.

4. Test the third query.
  - a. Rewrite the SQL statement that returns the same output using an IN-LINE VIEW. Use the following sample code given below or you can use the w5\_s4\_a.sql file.
    - The usage of both OR and ROWNUM sometimes causes performance problems. So, you rewrite the SQL statement that includes an IN-LINE VIEW.

```
SELECT cust_first_name, cust_last_name, cust_income_level,
cust_credit_limit
```

```
FROM (SELECT cust_first_name, cust_last_name, cust_income_level,
cust_credit_limit
FROM sh.cust
WHERE cust_income_level like 'F%' AND cust_state_province = 'NJ'
AND cust_gender= 'M'
union all
SELECT cust_first_name, cust_last_name, cust_income_level,
cust_credit_limit
FROM sh.cust
WHERE cust_income_level like 'F%' AND cust_state_province = 'NY'
AND cust_gender= 'M')
WHERE rownum < 31
/
```

**Note:** The inline view is a construct in Oracle SQL where you can place a query in the SQL FROM clause, as if the query was a table name.

- b. Execute the following sample code and gather the execution plan of the SQL statement by using the EXPLAIN PLAN command. You can use the w5\_s4\_b.sql file.

```
EXPLAIN PLAN FOR SELECT cust_first_name, cust_last_name,
cust_income_level, cust_credit_limit
FROM (SELECT cust_first_name, cust_last_name, cust_income_level,
cust_credit_limit
FROM sh.cust
WHERE cust_income_level like 'F%' AND cust_state_province = 'NJ'
AND cust_gender= 'M'
union all
SELECT cust_first_name, cust_last_name, cust_income_level,
cust_credit_limit
FROM sh.cust
WHERE cust_income_level like 'F%' AND cust_state_province = 'NY'
AND cust_gender= 'M')
WHERE rownum < 31
/
SELECT * FROM TABLE (dbms_xplan.display);
```

- 1) Which index was used?
- 2) What is the total cost?

The screenshot shows the SQL Worksheet interface in Oracle SQL Developer. The query being run is:

```

EXPLAIN PLAN FOR SELECT cust_first_name, cust_last_name, cust_income_level, cust_credit_limit
FROM (SELECT cust_first_name, cust_last_name, cust_income_level, cust_credit_limit
 FROM sh.cust
 WHERE cust_income_level like 'F%' AND cust_state_province = 'NJ'
 AND cust_gender= 'M'
 union all
 SELECT cust_first_name, cust_last_name, cust_income_level, cust_credit_limit
 FROM sh.cust
 WHERE cust_income_level like 'F%' AND cust_state_province = 'NY'
 AND cust_gender= 'M')
 WHERE rownum < 31
/
SELECT * FROM TABLE(dbms_xplan.display);

```

The Script Output window shows the execution completed in 0.049 seconds, indicating a successful plan. The PLAN\_TABLE\_OUTPUT section displays the execution plan details:

| Id | Operation                           | Name                  | Rows | Bytes | Cost (%CPU) | Time     |
|----|-------------------------------------|-----------------------|------|-------|-------------|----------|
| 0  | SELECT STATEMENT                    |                       | 30   | 1920  | 156 (0)     | 00:00:01 |
| 1* | COUNT STOPKEY                       |                       |      |       |             |          |
| 2  | VIEW                                |                       | 57   | 3648  | 156 (0)     | 00:00:01 |
| 3  | UNION-ALL                           |                       |      |       |             |          |
| 4* | TABLE ACCESS BY INDEX ROWID BATCHED | CUST                  | 27   | 1431  | 95 (0)      | 00:00:01 |
| 5* | INDEX RANGE SCAN                    | CUST_INCOME_STATE_IDX | 41   |       | 64 (0)      | 00:00:01 |
| 6* | TABLE ACCESS BY INDEX ROWID BATCHED | CUST                  | 30   | 1590  | 61 (0)      | 00:00:01 |
| 7* | INDEX RANGE SCAN                    | CUST_INCOME_STATE_IDX | 119  |       | 26 (0)      | 00:00:01 |

Predicate Information (identified by operation id):

- 1 - filter(ROWNUM<31)
- 4 - filter("CUST\_GENDER"='M')

### Answer:

- 1) The CUST\_INCOME\_STATE\_IDX index is used.
- 2) The cost is 156.

**Note:** The cost is 156 in this case. It might differ also.

- c. Compare the total costs of three SQL statements.
  - 1) Cost (Query 1)
  - 2) Cost (Query 2)
  - 3) Cost (Query 3)
  
5. Review the composite index.
  - a. You have tested three SQL statements that return the same output. All three SQL statements used the same index, which is CUST\_INCOME\_STATE\_IDX. You still feel that the performance could be improved. Now, you are investigating the CUST\_INCOME\_STATE\_IDX index for further improvement. Retrieve the order of columns in the composite index. You can use the w5\_s5\_a.sql file.

**Hint:** Use the USER\_IND\_COLUMNS view.

- 1) Column Name in Position 1: (\_\_\_\_\_)
- 2) Column Name in Position 2: (\_\_\_\_\_)

```
select COLUMN_NAME from USER_IND_COLUMNS where INDEX_NAME =
'CUST_INCOME_STATE_IDX';
```

The screenshot shows the Oracle SQL Developer interface. In the top tab bar, there are two tabs: 'w5\_s4\_b.sql' and 'w5\_s5\_asql'. The 'w5\_s5\_asql' tab is active. Below the tabs is a toolbar with various icons. The main area is divided into two panes: 'Worksheet' and 'Query Builder'. The 'Worksheet' pane contains the SQL query: 'select COLUMN\_NAME from USER\_IND\_COLUMNS where INDEX\_NAME = 'CUST\_INCOME\_STATE\_IDX''. The 'Query Result' pane below it displays the results of the query, which are two columns: 'COLUMN\_NAME' with values 'CUST\_INCOME\_LEVEL' and 'CUST\_STATE\_PROVINCE'.

| COLUMN_NAME           |
|-----------------------|
| 1 CUST_INCOME_LEVEL   |
| 2 CUST_STATE_PROVINCE |

**Answer:**

- 1) Column name in position 1 is `cust_income_level`.
  - 2) Column name in position 2 is `cust_state_province`.
- b. Check how these columns are used in the WHERE clause (Query 2).
- 1) WHERE (\_\_\_\_\_) LIKE '...'
  - 2) AND (\_\_\_\_\_) IN (....)

**Answer:**

- 1) WHERE `cust_income_level` LIKE 'F%'
  - 2) AND `cust_state_province` IN ('NJ', 'NY')
- c. Execute the following sample code and create a new composite index, `CUST_STATE_INCOME_IDX`, after considering the order of columns in the `CUST_INCOME_STATE_IDX` index for better performance. Make sure that you update statistics on the newly created index. You can use the `w5_s5_c.sql` file.

**Hint:** Think about the order of the Index column.

**Rule of thumb:** If all columns in a composite index are equally used, then consider the following:

- Point condition
- Distribution
- Access paths

```

DROP INDEX sh.cust_state_income_idx
/
CREATE INDEX sh.cust_state_income_idx
ON sh.cust (cust_state_province, cust_income_level);
NOLOGGING
COMPUTE STATISTICS
/
Exec DBMS_STATS_GATHER_TABLE_STATS ('SH','CUST', METHOD_OPT =>
'FOR ALL INDEXED COLUMNS', CASCADE =>TRUE)
/

```

SQL Worksheet History

Worksheet Query Builder

```

DROP INDEX sh.cust_state_income_idx
/
CREATE INDEX sh.cust_state_income_idx
ON sh.cust (cust_state_province, cust_income_level)
NOLOGGING
COMPUTE STATISTICS
/
exec DBMS_STATS.GATHER_TABLE_STATS('SH','CUST', METHOD_OPT => 'FOR ALL INDEXED COLUMNS', CASCADE => TRUE)
/

```

Script Output x

Task completed in 1.735 seconds

Error starting at line 1 in command:  
DROP INDEX sh.cust\_state\_income\_idx  
Error report:  
SQL Error: ORA-01418: specified index does not exist  
01418. 00000 - "specified index does not exist"  
\*Cause:  
\*Action:  
index SH.CUST\_STATE\_INCOME\_IDX created.  
anonymous block completed

- d. Execute the SQL statement (Query 2) and review the execution plan by using the sample code, or you can use the w5\_s5\_d.sql file.

```

EXPLAIN PLAN FOR SELECT cust_first_name, cust_last_name,
cust_income_level, cust_credit_limit
FROM sh.cust
WHERE cust_income_level like 'F%'
AND cust_state_province in ('NJ', 'NY')
AND cust_gender = 'M'
AND rownum < 31
/
SELECT * FROM TABLE (dbms_xplan.display);

```

- 1) Which index was used?

- 2) What is the total cost?

The screenshot shows the Oracle SQL Worksheet interface. In the 'Worksheet' tab, the following SQL code is displayed:

```

EXPLAIN PLAN FOR SELECT cust_first_name, cust_last_name, cust_income_level, cust_credit_limit
FROM sh.cust
WHERE cust_income_level like 'F%'
AND cust_state_province in ('NJ', 'NY')
AND cust_gender = 'M'
AND rownum < 31
/
SELECT * FROM TABLE(dbms_xplan.display);

```

In the 'Script Output' tab, the results of the EXPLAIN PLAN are shown:

```

plan FOR succeeded.
PLAN_TABLE_OUTPUT

Plan hash value: 1371959703

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time |
|---|---|---|---|---|---|---|
| 0 | SELECT STATEMENT | | 30 | 1590 | 38 (0) | 00:00:01 |
|* 1 | COUNT STOPKEY | | | | | |
| 2 | INLIST ITERATOR | | | | | |
|* 3 | TABLE ACCESS BY INDEX ROWID BATCHED | CUST | 30 | 1590 | 38 (0) | 00:00:01 |
|* 4 | INDEX RANGE SCAN | CUST_STATE_INCOME_IDX | 169 | | 3 (0) | 00:00:01 |

Predicate Information (identified by operation id):

1 - filter(ROWNUM<31)
3 - filter("CUST_GENDER"='M')
4 - access(("CUST_STATE_PROVINCE"='NJ' OR "CUST_STATE_PROVINCE"='NY') AND "CUST_INCOME_LEVEL" LIKE 'F%')
 filter("CUST_INCOME_LEVEL" LIKE 'F%')

20 rows selected

```

**Answer:**

- 1) The CUST\_STATE\_INCOME\_IDX index is used.
- 2) The cost is 38.

**Note:** The cost is 38 in this case. It might differ also.

- e. Did you observe a better cost?

**Answer:** Yes, the cost is better.

- f. Clean up the environment. Use the cleanup05.sql file.

```

DROP INDEX sh.cust_state_idx
/
DROP INDEX sh.cust_state_gender_idx
/
DROP INDEX sh.cust_income_state_idx

```

```
/
DROP INDEX sh.cust_state_income_idx
/
```

6. Which statements are true regarding guidelines for ordering keys in composite indexes?
  - a. Create the index so that the keys used in WHERE clauses make up a leading portion.
  - b. If some keys appear in WHERE clauses more frequently, then create the index so that the more frequently selected keys make up a leading portion in order to allow the statements that use only these keys to use the index.
  - c. If all keys appear in WHERE clauses equally often but the data is physically ordered on one of the keys, then place this key first in the composite index.

**Answer:** a, b, and c

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Error : You are not a Valid Partner use only

# **Workshop 6**

## **Chapter 19**

## **Workshop 6: Overview**

---

### **Workshop Overview**

In this workshop, you will perform the following tasks:

- Review the execution plan and several sections in an event 10053 trace file.
- Interpret the information to understand the optimizer's decision.
- Learn how to use the information in the 10053 file to tune the SQL statement.

## Workshop-6: Using Information in the 10053 File to Tune a SQL Statement

### Overview

In this workshop, you review the following execution plan and several sections in an event 10053 trace file. There are also several questions on cost model, selectivity, and cardinality. You interpret the information to understand the optimizer's decision. Finally, you learn how to verify the cost of a nested loops manually. Note that this workshop is only for demonstration purposes.

```
SELECT ename, e.deptno, d.deptno, d.dname
FROM emp e, dept d
WHERE e.deptno = d.deptno and ename like 'A%'
/
```

| Id  | Operation                   | Name    | Rows | Bytes | Cost (%CPU) | Time |
|-----|-----------------------------|---------|------|-------|-------------|------|
| 0   | SELECT STATEMENT            |         | 1    | 22    | 4           | (0)  |
| 1   | NESTED LOOPS                |         |      |       |             |      |
| 2   | NESTED LOOPS                |         | 1    | 22    | 4           | (0)  |
| * 3 | TABLE ACCESS FULL           | EMP     | 1    | 9     | 3           | (0)  |
| * 4 | INDEX UNIQUE SCAN           | PK_DEPT | 1    |       | 0           | (0)  |
| 5   | TABLE ACCESS BY INDEX ROWID | DEPT    | 1    | 13    | 1           | (0)  |

### Tasks

- From where/how is the cost of the simple nested loops 4 derived?
  - First of all, review the information needed to understand the CBO cost model:
  - a. Cost = IO cost + CPU cost
  - b. sreadtim = ioseektim + db\_block\_size / iotfrspeed
  - c. mreadtim = ioseektim + db\_file\_multiblock\_read\_count \* db\_block\_size / iotfrspeed
  - d. CPU Costing Factor = cpuspeed \* 1000 \* sreadtim
  - e. CPU Cost Scale Factor = 1 / CPU Costing Factor
  - f. CPU Cost = Resp\_cpu \* CPU Cost Scale Factor
  - g. NL Cost = Outer Table Cost + (Outer Table Card \* Inner Table Cost)
  - h. HJ/SMJ Cost = Outer Table Cost + Inner Table Cost + Join Cost2

2. Review the SYSTEM STATISTICS section in an event 10053 trace file and calculate the following information:
- sreadtim (db\_block\_size: 8K): (\_\_\_\_\_)
  - CPU cost scale factor: (\_\_\_\_\_)

```

SYSTEM STATISTICS INFORMATION

Using NOWORKLOAD Stats
CPUSPEEDNW: 1999 millions instructions/sec (default is 100)
IOTFRSPEED: 4096 bytes per millisecond (default is 4096)
IOSEEKTIM: 10 milliseconds (default is 10)
MBRC: -1 blocks (default is 8)
```

$$\begin{aligned} \text{sreadtim} &= \text{ioseektim} + \text{db\_block\_size} / \text{iotfrspeed} \\ &= 10\text{ms} + 8096/4096 = 12 \text{ ms} \\ \text{CPU Costing Factor} &= \text{cpuspeed} * 1000 * \text{sreadtim} \\ &= 1999 * 1000 * 12 = 23988000 \\ \text{CPU Cost Scale Factor} &= 1 / \text{CPU Costing Factor} \\ &= 1/(1999 * 1000 * 12) = 0.000000417 (\text{nearly zero}) \end{aligned}$$

3. Review the following section for the outer table and where is the outer table cost 3 from?
- Outer Table Cost (\_\_\_) = IO Cost (\_\_\_\_) + CPU Cost (\_\_\_\_)

```
Access path analysis for EMP

SINGLE TABLE ACCESS PATH
Single Table Cardinality Estimation for EMP[E]
Column (#2): ENAME(
 AvgLen: 6 NDV: 14 Nulls: 0 Density: 0.071429
Table: EMP Alias: E
 Card: Original: 14.000000 Rounded: 1 Computed: 1.00 Non Adjusted: 1.00
Access Path: TableScan
 Cost: 3.00 Resp: 3.00 Degree: 0
 Cost io: 3.00 Cost cpu: 39507
 Resp_io: 3.00 Resp_cpu: 39507
Best:: AccessPath: TableScan
 Cost: 3.00 Degree: 1 Resp: 3.00 Card: 1.00 Bytes: 0
```

$$\begin{aligned} \text{Outer Table Cost} &= \text{IO Cost} + \text{CPU Cost} \\ &= \text{Resp\_io} + (\text{Resp\_cpu} * \text{CPU Cost Scale Factor}) \\ &= 3 + (39507 * 0.000000417) \\ &= 3 + 0.0162 (\text{nearly zero}) \end{aligned}$$

4. Review the following section that includes the inner table information.
- Outer Table Card: (\_\_\_\_\_)
  - Inner Table Cost (\_\_\_\_\_) = IO Cost (\_\_\_\_) + CPU Cost (\_\_\_\_\_)

```

Now joining: DEPT[D]#1

NL Join
Outer table: Card: 1.00 Cost: 3.00 Resp: 3.00 Degree: 1 Bytes: 9
Access path analysis for DEPT
 Inner table: DEPT Alias: D
 Access Path: TableScan
 NL Join: Cost: 6.00 Resp: 6.00 Degree: 1
 Cost_io: 6.00 Cost_cpu: 75794
 Resp io: 6.00 Resp cpu: 75794
 Access Path: index (UniqueScan)
 Index: PK_DEPT
 resc_io: 1.00 resc_cpu: 8341
 ix_sel: 0.250000 ix_sel_with_filters: 0.250000
 NL Join : Cost: 4.00 Resp: 4.00 Degree: 1
 Cost_io: 4.00 Cost_cpu: 47849
 Resp io: 4.00 Resp cpu: 47849

(Output truncated)

Best NL cost: 4.00
 resc: 4.00 resc_io: 4.00 resc_cpu: 47849
 resp: 4.00 resp_io: 4.00 resp_cpu: 47849
Join Card: 1.000000 = = outer (1.000000) * inner (4.000000) * sel (0.250000)
Join Card - Rounded: 1 Computed: 1.00
```

$$\begin{aligned}
 \text{Inner Table Cost} &= \text{IO Cost} + \text{CPU Cost} \\
 &= \text{resc\_io} + (\text{resc\_cpu} * \text{CPU Cost Scale Factor}) \\
 &= 1 + (8341 * 0.000000417) \\
 &= 1 + 0.0162 \text{ (nearly zero)}
 \end{aligned}$$

5. Where is the cost of the nested loops from?

- NL Cost = Outer Table Cost + (Outer Table Card \* Inner Table Cost)
- $4 = 3 + (1 * 1)$

6. Which statements regarding nested loops joins are TRUE?
- a. The optimizer uses nested loops joins when joining a small number of rows, with a good driving condition between the two tables.
  - b. The cost of nested loops is easy to verify manually by using an event 10053.
  - c. A major component of nested loops cost is CPU cost.
  - d. A major component of nested loops cost is I/O cost.

**Answer:** a, b, d

## **Workshop 7**

### **Chapter 20**

## **Workshop 7: Overview**

---

### **Workshop Overview**

In this workshop, you will perform the following tasks:

- Review the execution plan and several sections in an event 10053 trace file.
- Interpret the information to understand the optimizer's decision.
- Learn how to use the information in the 10053 file to tune a SQL statement.

## Workshop-7: Understanding the Optimizer's Decision

### Overview

In this workshop, you review the following execution plan and several sections in an event 10053 trace file. You interpret the information to understand the optimizer's decision. Note that this workshop is only for demonstration purposes.

```

SELECT ch.channel_class, c.cust_city,
 t.calendar_quarter_desc,
 SUM(s.amount_sold) sales_amount
 FROM sales s,times t,customers c,channels ch
 WHERE s.time_id = t.time_id AND
 s.cust_id = c.cust_id AND
 s.channel_id = ch.channel_id AND
 c.cust_state_province = 'CA' AND
 ch.channel_desc IN ('Internet','Catalog') AND
 t.calendar_quarter_desc IN ('1999-Q1','1999-Q2')
 GROUP BY ch.channel_class, c.cust_city,
 t.calendar_quarter_desc
/

```

| Id  | Operation                   | Name      | Rows  | Bytes | Cost (%CPU) | Time | Pstart   | Pstop           |
|-----|-----------------------------|-----------|-------|-------|-------------|------|----------|-----------------|
| * 0 | SELECT STATEMENT            |           | 572   | 48048 | 962         | (2)  | 00:00:12 |                 |
| * 1 | HASH GROUP BY               |           | 572   | 48048 | 962         | (2)  | 00:00:12 |                 |
| * 2 | HASH JOIN                   |           | 3116  | 255K  | 961         | (2)  | 00:00:12 |                 |
| * 3 | TABLE ACCESS FULL           | CHANNELS  | 2     | 42    | 3           | (0)  | 00:00:01 |                 |
| * 4 | HASH JOIN                   |           | 6231  | 383K  | 957         | (2)  | 00:00:12 |                 |
| 5   | PART JOIN FILTER CREATE     | :BF0000   | 183   | 2928  | 18          | (0)  | 00:00:01 |                 |
| * 6 | TABLE ACCESS FULL           | TIMES     | 183   | 2928  | 18          | (0)  | 00:00:01 |                 |
| * 7 | HASH JOIN                   |           | 49822 | 2286K | 939         | (2)  | 00:00:12 |                 |
| * 8 | TABLE ACCESS FULL           | CUSTOMERS | 383   | 9958  | 405         | (1)  | 00:00:05 |                 |
| 9   | PARTITION RANGE JOIN-FILTER |           | 918K  | 18M   | 529         | (3)  | 00:00:07 | :BF0000 :BF0000 |
| 10  | TABLE ACCESS FULL           | SALES     | 918K  | 18M   | 529         | (3)  | 00:00:07 | :BF0000 :BF0000 |

### Tasks

1. Review the preceding execution plan.
  - a. What is the plan cost? (\_\_\_\_\_)
  - b. Where does the 00:00:12 time come from? (\_\_\_\_\_)

### Answer:

- a. Plan cost is 962.
  - b.  $12 \text{ sec} = \text{Plan Cost} * \text{sreadtim} = 962 * 12\text{ms}$
2. What is the meaning of a cost of 962 and 00:00:12 time.

**Note:** The meaning of the cost 962 is that the Oracle Optimizer computed that the above query should take as long as 962 single block reads. The cost of 962 equals an elapsed time of 12 sec.

3. Review the following section in an event 10053. What should be checked here?

```

PARAMETERS USED BY THE OPTIMIZER

PARAMETERS WITH ALTERED VALUES

Compilation Environment Dump

*** 2011-06-27 16:49:55.273
_db_file_optimizer_read_count = 16
cursor_sharing = force
db_file_multiblock_read_count = 16
Bug Fix Control Environment

PARAMETERS WITH DEFAULT VALUES

Compilation Environment Dump
optimizer_mode_hinted = false
optimizer_features_hinted = 0.0.0
parallel_execution_enabled = true
```

**Note:** You have learned in the lesson titled “Optimizer Fundamentals” that there are several parameters that could control the behavior of the optimizer.

The example in 10053 shows parameters used by the optimizer.

While reviewing this section, you can examine if any altered parameter values influenced the optimizer’s decision.

You can also compare 10053 traces between two versions or two different plans to see why the plan cost is different.

4. Review the following section in an event 10053. What should be checked here?

```

BASE STATISTICAL INFORMATION

Table Stats:::
 Table: CHANNELS Alias: CH
 #Rows: 5 #Blks: 4 AvgRowLen: 40.00
Index Stats:::
 Index: CHANNELS_PK Col#: 1
 █ LVLS: 0 #LB: 1 #DK: 5 LB/K: 1.00 DB/K: 1.00 CLUF: 1.00

Table Stats:::
 Table: CUSTOMERS Alias: C
 #Rows: 55500 #Blks: 1486 AvgRowLen: 180.00
Index Stats:::
 Index: CUSTOMERS_GENDER_BIX Col#: 4
 █ LVLS: 1 #LB: 3 #DK: 2 LB/K: 1.00 DB/K: 2.00 CLUF: 5.00
 Index: CUSTOMERS_MARITAL_BIX Col#: 6
```

**Note:** It shows statistics gathered for each table and index referred by the query.

Look for any tables that were not analyzed. You see “NOT ANALYZED” in the table stats section if there is no analysis. Indexes that are not analyzed can be identified by default.

5. Review the following section in an event 10053. What should be checked here?

```
Access path analysis for SALES

SINGLE TABLE ACCESS PATH
 Single Table Cardinality Estimation for SALES[S]
 Table: SALES Alias: S
 Card: Original: 918843.000000 Rounded: 918843 Computed: 918843.00 Non Adjusted
918843.00
 Access Path: TableScan
 Cost: 405.99 Resp: 405.99 Degree: 0
 Cost_io: 389.00 Cost_cpu: 260685437
 Resp_io: 389.00 Resp_cpu: 260685437
 ***** trying bitmap/domain indexes *****
 Access Path: index (FullScan)
 Index: SALES_CHANNEL_BIX
 resc_io: 75.00 resc_cpu: 552508
 ix_sel: 1.000000 ix_sel_with_filters: 1.000000
 Cost: 75.04 Resp: 75.04 Degree: 0
 Access Path: index (FullScan)
 Index: SALES_CUST_BIX
 resc_io: 503.00 resc_cpu: 10743684
 ix_sel: 1.000000 ix_sel_with_filters: 1.000000
 Cost: 503.70 Resp: 503.70 Degree: 0
 Access Path: index (FullScan)
 Index: SALES_PROD_BIX
```

```
Best:: AccessPath: TableScan
 Cost: 405.99 Degree: 1 Resp: 405.99 Card: 918843.00 Bytes: 0
```

**Note:** Single table access path – You can determine the path that correlates to the execution plan. At the end of this section, you can see the best single table access cost.

- Review the following section in an event 10053. What should be checked here?

```

OPTIMIZER STATISTICS AND COMPUTATIONS

GENERAL PLANS

Considering cardinality-based initial join order.
Permutations for Starting Table :0
Join order[1]: CHANNELS[CH]#0 TIMES[T]#1 CUSTOMERS[C]#2 SALES[S]#3

Now joining: TIMES[T]#1

NL Join
Outer table: Card: 2.00 Cost: 3.00 Resp: 3.00 Degree: 1 Bytes: 21
Access path analysis for TIMES
Inner table: TIMES Alias: T
Access Path: TableScan
NL Join: Cost: 31.30 Resp: 31.30 Degree: 1
Cost_io: 31.00 Cost_cpu: 4659106
Resp_io: 31.00 Resp_cpu: 4659106

```

**Note:** Join orders and join methods

- Suppose that a user is complaining of slow response time of a query. According to the user's observation, it took 20 minutes to complete. You reviewed the execution plan and cost by using the explain plan and plan\_table. The plan cost is 4000.
  - What is the computed elapsed time based on the cost of 4000?
  - Was the Oracle Optimizer estimated accurately?
    - Yes
    - No
  - What could be the possible reasons for an inaccurate cost?

**Answer:**

- Computed elapsed time = Plan Cost \* sreadtim (12 ms)  
 $= 4000 \times 12(\text{ms})$   
 $= 48000 \text{ ms}$
- No
- The reasons for inaccurate cost are:
  - Large IN list
  - Large OR statement

- 3) Different estimated cardinality
  - 4) Incorrect selectivity estimate
8. Which statements are TRUE regarding hash joins?
- a. System statistics can influence the optimizer's decision because hash joins require the intense use of memory and CPU.
  - b. Hash joins are performed for equijoins, non-equijoins, and are most useful when joining large amounts of data.
  - c. Hash joins can take advantage of the index on the join key column.

**Answer:** a

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Error : You are not a Valid Partner use only

# **Workshop 8**

## **Chapter 21**

## Workshop 8: Overview

---

### Workshop Overview

In this workshop, you will perform the following tasks:

- Review parse time reduction strategy.
- Test without cursor sharing.
- Use CURSOR\_SHARING.
- Use user-defined bind variables.
- Load SQL statements into a SQL plan baseline.

### Scripts used in the Workshop

- `ws08.sql`
- `w8_s2_c.sql`
- `w8_s2_d.sql`
- `w8_s2_e.sql`
- `w8_s2_f.sql`
- `w8_s2_g.sql`
- `w8_s3_a.sql`
- `w8_s3_c.sql`
- `w8_s3_c1.sql`
- `w8_s3_d.sql`
- `w8_s4_b.sql`
- `w8_s4_c.sql`
- `w8_s5_a.sql`
- `w8_s5_a1.sql`
- `w8_s5_b.sql`

## Workshop 8: Tuning Strategy

---

### Overview

You have identified a slow-running SQL statement. It was run 555 times in a certain time period. You noticed that a different literal was used in each execution. This caused the system to parse the same statement 555 times with the high parse time, which is not efficient. What tuning strategy could be used?

| call    | count | cpu    | elapsed | disk | query   | current | rows  |
|---------|-------|--------|---------|------|---------|---------|-------|
| Parse   | 555   | 100.09 | 300.83  | 0    | 0       | 0       | 0     |
| Execute | 555   | 0.42   | 0.78    | 0    | 0       | 0       | 0     |
| Fetch   | 555   | 14.04  | 85.03   | 513  | 1448514 | 0       | 11724 |
| total   | 1665  | 114.55 | 386.65  | 513  | 1448514 | 0       | 11724 |

### Tasks

1. Review parse time reduction strategy.
  - a. Review the output of TKPROF above and provide your answers for the following questions.
    - 1) What is the elapsed time spent parsing? (\_\_\_\_\_).
    - 2) What is the elapsed time spent executing? (\_\_\_\_\_)
    - 3) What is the elapsed time spent fetching? (\_\_\_\_\_)
    - 4) Would normal query tuning techniques that alter the execution plan help?
    - 5) Would reducing parse time help? (**Note:** Assume that there is no a client network bottleneck issue.)

#### Answer:

- 1) 300.83
- 2) 0.78
- 3) 85.03
- 4) No
- 5) Parse time reduction strategy can be used when you have determined that the query spends most of its time in the parse phase.

- b. How would you reduce the parse time? You could consider the following two options:

**Possible Option 1:** Minimize hard parses.

- 1) Cursor sharing using bind variables
- 2) Adaptive cursor sharing

**Possible Option 2:** Reduce the parse elapsed time.

- a. If CPU time dominates the parse time, check the following common observations, and others, and apply solutions to resolve the issue.
    - 1) Dynamic sampling is being used for the query and impacting the parse time
    - 2) Query has many IN-list parameters
    - 3) Query has many OR operators
    - 4) Partitioned table with many partitions (for example, more than 1000)
  - b. If wait time dominates the parse time, check the following common observations, and others, and apply solutions to resolve the issue.  
Waits for large query text to be sent from the client
  - c. In this workshop, you choose option 1 to reduce the number of hard parses, eventually to reduce the total parse time.
- 
2. Test without cursor sharing.
    - a. Connect to sh schema in SQL Developer.
    - b. Review and execute the SQL statements in ws08.sql. You consider the use of bind variables to reduce the parse time.

**Note:** The scripts used in this workshop are available in the workshops folder located in /home/oracle/labs/workshops.

```
SELECT cust_city, sum (amount_sold)
FROM sales s, customers c
WHERE s.cust_id=c.cust_id and country_id = 52790
GROUP BY cust_city
ORDER BY 2;
SELECT cust_city, sum (amount_sold)
FROM sales s, customers c
WHERE s.cust_id=c.cust_id and country_id = 52787
GROUP BY cust_city
ORDER BY 2;
```

- 1) Which column is a good candidate for a bind variable? (\_\_\_\_\_)

**Answer:** country\_id

- c. Determine the distribution of all distinct values found in the candidate column in step b.

You can use the w8\_s2\_c.sql file.

SQL Worksheet History

Worksheet Query Builder

```
SELECT country_id, count(*), (count(*)*100)/55500 PCT
FROM sh.customers
GROUP BY country_id
ORDER BY country_id
```

Query Result | All Rows Fetched: 19 in 0.025 seconds

|    | COUNTRY_ID | COUNT(*) | PCT                                            |
|----|------------|----------|------------------------------------------------|
| 1  | 52769      | 597      | 1.07567567567567567567567567567567567567567568 |
| 2  | 52770      | 7780     | 14.01801801801801801801801801801801801802      |
| 3  | 52771      | 712      | 1.28288288288288288288288288288288288288       |
| 4  | 52772      | 2010     | 3.62162162162162162162162162162162162162162    |
| 5  | 52773      | 403      | 0.7261261261261261261261261261261261261261     |
| 6  | 52774      | 831      | 1.4972972972972972972972972972972972972973     |
| 7  | 52775      | 832      | 1.4990990990990990990990990990990990990991     |
| 8  | 52776      | 8173     | 14.72612612612612612612612612612612612613      |
| 9  | 52777      | 383      | 0.6900900900900900900900900900900900900901     |
| 10 | 52778      | 2039     | 3.67387387387387387387387387387387387          |
| 11 | 52779      | 3833     | 6.90630630630630630630630630630630630631       |
| 12 | 52782      | 624      | 1.12432432432432432432432432432432432432       |
| 13 | 52785      | 244      | 0.4396396396396396396396396396396396396        |
| 14 | 52786      | 708      | 1.27567567567567567567567567567567567568       |
| 15 | 52787      | 75       | 0.1351351351351351351351351351351351351        |
| 16 | 52788      | 91       | 0.163963963963963963963963963963963963964      |
| 17 | 52789      | 7557     | 13.61621621621621621621621621621621622         |
| 18 | 52790      | 18520    | 33.36936936936936936936936936936936936936937   |
| 19 | 52791      | 88       | 0.158558558558558558558558558558558558558586   |

- d. If needed, create a histogram on the column and check it by using the USER\_TAB\_COLUMNS view. You can use the w8\_s2\_d.sql file.

  - 1) What is the histogram type? (\_\_\_\_\_)
  - 2) How many buckets exist in the histogram? (\_\_\_\_\_)

```
exec DBMS_STATS.GATHER_TABLE_STATS(OWNNAME=>'SH',
TABNAME=>'CUSTOMERS', METHOD_OPT => 'FOR COLUMNS SIZE 25
country id', CASCADE => TRUE)
```

```
SELECT column_name, histogram, num_buckets
FROM user_tab_columns
WHERE table_name = 'CUSTOMERS'
/
```

|    | COLUMN_NAME            | HISTOGRAM | NUM_BUCKETS |
|----|------------------------|-----------|-------------|
| 1  | CUST_VALID             | FREQUENCY | 2           |
| 2  | CUST_EFF_TO            | NONE      | 0           |
| 3  | CUST_EFF_FROM          | FREQUENCY | 1           |
| 4  | CUST_SRC_ID            | NONE      | 0           |
| 5  | CUST_TOTAL_ID          | FREQUENCY | 1           |
| 6  | CUST_TOTAL             | FREQUENCY | 1           |
| 7  | CUST_EMAIL             | HYBRID    | 254         |
| 8  | CUST_CREDIT_LIMIT      | FREQUENCY | 8           |
| 9  | CUST_INCOME_LEVEL      | FREQUENCY | 12          |
| 10 | CUST_MAIN_PHONE_NUMBER | HYBRID    | 254         |
| 11 | COUNTRY_ID             | FREQUENCY | 19          |
| 12 | CUST_STATE_PROVINCE_ID | FREQUENCY | 145         |
| 13 | CUST_STATE_PROVINCE    | FREQUENCY | 145         |
| 14 | CUST_CITY_ID           | HYBRID    | 254         |
| 15 | CUST_CITY              | HYBRID    | 254         |
| 16 | CUST_POSTAL_CODE       | HYBRID    | 254         |
| 17 | CUST_STREET_ADDRESS    | HYBRID    | 254         |
| 18 | CUST_MARITAL_STATUS    | FREQUENCY | 11          |
| 19 | CUST_YEAR_OF_BIRTH     | FREQUENCY | 75          |

**Answer:**

- 1) A frequency histogram is used.
  - 2) Number of buckets for COUNTRY\_ID: 19. It means there are only 19 distinct values in the column.
- e. Create an index on the column identified in step d by using the following sample code, or use the w8\_s2\_e.sql file.

```
DROP INDEX sh.cust_country_id_idx
/
CREATE INDEX sh.cust_country_id_idx
ON sh.customers (country_id)
/
```

The screenshot shows an Oracle SQL Worksheet interface. The top tab bar has two tabs: 'ws08.sql' and 'w8\_s2\_e.sql'. The 'w8\_s2\_e.sql' tab is active. The main workspace contains the following SQL code:

```
DROP INDEX sh.cust_country_id_idx
/
CREATE INDEX sh.cust_country_id_idx
ON sh.customers(country_id)
/
```

Below the workspace is a 'Script Output' window. It displays the following error message:

```
Error starting at line 1 in command:
DROP INDEX sh.cust_country_id_idx
Error report:
SQL Error: ORA-01418: specified index does not exist
01418. 00000 - "specified index does not exist"
*Cause:
*Action:
index SH.CUST_COUNTRY_ID_IDX created.
```

- f. Determine the execution plan of the following SQL statement. You can use the w8\_s2\_f.sql file.

```
SELECT cust_city, sum(amount_sold)
FROM sales s, customers c
WHERE s.cust_id=c.cust_id and country_id = 52790
GROUP BY cust_city
ORDER BY 2
/
SELECT * FROM table(dbms_xplan.display_cursor);
```

The screenshot shows the Oracle SQL Worksheet interface. The top window is titled "ws08.sql x w8\_s2\_f.sql x" and contains the following SQL code:

```

SELECT cust_city,sum(amount_sold)
 FROM sales s, customers c
 WHERE s.cust_id=c.cust_id and country_id = 52790
 GROUP BY cust_city
 ORDER BY 2

SELECT * FROM table (dbms_xplan.display_cursor);

```

The bottom window is titled "Script Output x" and displays the execution plan:

```

Plan hash value: 947950142

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time | Pstart | Pstop |
|---|---|---|---|---|---|---|---|---|
| 0 | SELECT STATEMENT | | | | 983 (100) | | | |
| 1 | SORT ORDER BY | | 620 | 18600 | 983 (6) | 00:00:01 | | |
| 2 | HASH GROUP BY | | 620 | 18600 | 983 (6) | 00:00:01 | | |
|* 3 | HASH JOIN | | 918K | 26M | 942 (2) | 00:00:01 | | |
|* 4 | TABLE ACCESS FULL| CUSTOMERS | 18520 | 361K | 423 (1) | 00:00:01 | | |
| 5 | PARTITION RANGE ALL| | 918K | 8973K | 517 (2) | 00:00:01 | 1 | 28 |
| 6 | TABLE ACCESS FULL| SALES | 918K | 8973K | 517 (2) | 00:00:01 | 1 | 28 |

Predicate Information (identified by operation id):

 3 - access("S"."CUST_ID"="C"."CUST_ID")
 4 - filter("COUNTRY_ID=52790")

Note

- this is an adaptive plan

30 rows selected

```

**Note:** 33.3% of the duplicated data is present in the COUNTRY\_ID column where the value of country\_id = 52790.

- g. Determine the execution plan of the following SQL statement. You can use the w8\_s2\_g.sql file.

```

SELECT cust_city, sum (amount_sold)
 FROM sales s, customers c
 WHERE s.cust_id=c.cust_id and country_id = 52787
 GROUP BY cust_city
 ORDER BY 2
/
SELECT * FROM table (dbms_xplan.display_cursor);

```

```

SELECT cust_city,sum(amount_sold)
 FROM sales s, customers c
 WHERE s.cust_id=c.cust_id and country_id = 52787
 GROUP BY cust_city
 ORDER BY 2

SELECT * FROM table(dbms_xplan.display_cursor);

```

Plan hash value: 2854028564

| Id  | Operation                           | Name                | Rows | Bytes | Cost (%CPU) | Time     | Pstart | Pstop |
|-----|-------------------------------------|---------------------|------|-------|-------------|----------|--------|-------|
| 0   | SELECT STATEMENT                    |                     |      |       | 537 (100)   |          |        |       |
| 1   | SORT ORDER BY                       |                     | 71   | 2130  | 537 (2)     | 00:00:01 |        |       |
| 2   | HASH GROUP BY                       |                     | 71   | 2130  | 537 (2)     | 00:00:01 |        |       |
| * 3 | HASH JOIN                           |                     | 9762 | 285K  | 537 (2)     | 00:00:01 |        |       |
| 4   | TABLE ACCESS BY INDEX ROWID BATCHED | CUSTOMERS           | 75   | 1500  | 18 (0)      | 00:00:01 |        |       |
| * 5 | INDEX RANGE SCAN                    | CUST_COUNTRY_ID_IDX | 75   |       | 1 (0)       | 00:00:01 |        |       |
| 6   | PARTITION RANGE ALL                 |                     | 918K | 3973K | 517 (2)     | 00:00:01 | 1      | 28    |
| 7   | TABLE ACCESS FULL                   | SALES               | 918K | 3973K | 517 (2)     | 00:00:01 | 1      | 28    |

Predicate Information (identified by operation id):

- 3 - access("S"."CUST\_ID"="C"."CUST\_ID")
- 5 - access("COUNTRY\_ID"=52787)

Note

-----

- this is an adaptive plan

31 rows selected

**Note:** 0.135% of the duplicated data is present in the COUNTRY\_ID column where the value of country\_id = 52787. In this plan, CUST\_COUNTRY\_ID\_IDX is used.

- Try to explain why the execution plans are different in step f and g.
    - Why are the plans different? Does the optimizer consider two similar SQL statements in step f and g as the same SQL statements?
    - How many hard parses occurred?
  - Reduce the number of hard parses in Task 3.
3. Use CURSOR\_SHARING
- Assume that the identified SQL statements in ws08.sql are all hard-coded. That means that you cannot use any user-defined bind variables because the SQL statements cannot be modified. In this task, you use system-level bind variable instead.
- Note:** In this workshop, you will choose option 1 to reduce the number of hard parses, eventually to reduce the total parse time.
- Remember that the optimizer assigns a value to a bind variable only in the first invocation to generate a good execution plan. The same execution plan is used for the subsequent SQL

statements as if the plan is the optimal plan in all cases. In general, this behavior is good in most cases. But it would not work if a bind variable column has a skewed data distribution and even with histograms. A histogram is used only for the peeked value in the first invocation due to bind variable peeking. You know that a bind variable does not work well with a histogram.

You have learned about adaptive cursor sharing in the lesson titled “Introduction to Optimizer Statistics Concepts”. You will examine if adaptive cursor sharing can help the optimizer share a cursor safely and in an intelligent way instead of sharing a cursor blindly.

- a. Use the following sample trigger to create a LOGON trigger. **You have to set the cursor\_sharing parameter to a proper value for system bind variables as soon as sh user logs on.**

Connect as user sys and execute the sample code or you can use the w8\_s3\_a.sql file. Connection details are as follows:

Connection Name: sys\_connection

Username: sys

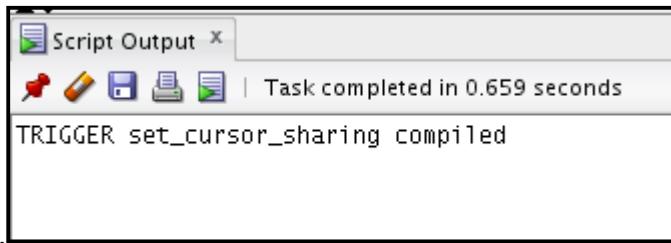
Password: oracle\_4U

Role: SYSDBA

Host Name: localhost

SID: systun

```
create or replace trigger set_cursor_sharing
after logon
on database
declare
begin
 if user = 'SH' then execute immediate 'alter session
set cursor_sharing=<value>';
 end if;
exception
 when others then
 null;
end;
/
```



- b. Connect to the sh schema in SQL Developer.

- c. Execute the following SQL statements to verify whether the system bind variable was used. You can use the `w8_s3_c.sql` and `w8_s3_c1.sql` files.

```
SELECT /* TASK04 */ cust_city, sum (amount_sold)
FROM sales s, customers c
WHERE s.cust_id=c.cust_id and country_id = 52790
GROUP BY cust_city
ORDER BY 2
/
SELECT * FROM table (dbms_xplan.display_cursor)
/
```

```
SELECT /* TASK04 */ cust_city, sum (amount_sold)
FROM sales s, customers c
WHERE s.cust_id=c.cust_id and country_id = 52787
GROUP BY cust_city
ORDER BY 2
/
SELECT * FROM table (dbms_xplan.display_cursor)
/
```

The screenshot shows the Oracle SQL Worksheet interface. At the top, there are tabs for 'ws08.sql', 'w8\_s2\_f.sql', 'system\_conn', and 'w8\_s3\_c.sql'. The current tab is 'w8\_s3\_c.sql'. The title bar indicates 'SQL Worksheet History' and 'sh\_conn'. The main area contains the following SQL code:

```
/* TASK04 */ SELECT /* TASK04 */ cust_city, sum(amount_sold)
FROM sales s, customers c
WHERE s.cust_id=c.cust_id and country_id = 52790
GROUP BY cust_city
ORDER BY 2
/
SELECT * FROM table(dbms_xplan.display_cursor)
```

Below the code, the 'Query Result' tab is active, showing the output of the query:

| Customer City | Total Amount Sold |
|---------------|-------------------|
| San Mateo     | 2395899.7         |

The message '233 rows selected' is displayed below the table.

The 'PLAN\_TABLE\_OUTPUT' section shows the execution plan:

```
SQL_ID 98npy7u9rt7s7, child number 0
SELECT /* TASK04 */ cust_city, sum(amount_sold) FROM sales s, customers
c WHERE s.cust_id=c.cust_id and country_id = 52790 GROUP BY cust_city
ORDER BY 2
```

The 'Plan hash value: 947950142' is also shown.

The execution plan table is as follows:

| Id | Operation           | Name              | Rows      | Bytes | Cost (%CPU) | Time     | Pstart   | Pstop |
|----|---------------------|-------------------|-----------|-------|-------------|----------|----------|-------|
| 0  | SELECT STATEMENT    |                   |           |       | 983 (100)   |          |          |       |
| 1  | SORT ORDER BY       |                   | 620       | 18600 | 983 (6)     | 00:00:01 |          |       |
| 2  | HASH GROUP BY       |                   | 620       | 18600 | 983 (6)     | 00:00:01 |          |       |
| *  | HASH JOIN           |                   | 918K      | 26M   | 942 (2)     | 00:00:01 |          |       |
| *  | 4                   | TABLE ACCESS FULL | CUSTOMERS | 18520 | 361K        | 423 (1)  | 00:00:01 |       |
| 5  | PARTITION RANGE ALL |                   | 918K      | 8973K | 517 (2)     | 00:00:01 | 1        | 28    |
| 6  | TABLE ACCESS FULL   | SALES             | 918K      | 8973K | 517 (2)     | 00:00:01 | 1        | 28    |

The 'Predicate Information (identified by operation id):' section shows:

```
3 - access("S"."CUST_ID"="C"."CUST_ID")
```

The screenshot shows the Oracle SQL Developer interface. In the top tab bar, three tabs are open: 'ws08.sql', 'w8\_s3\_c.sql', and 'w8\_s3\_cLsql'. The 'w8\_s3\_cLsql' tab is active, showing a SQL Worksheet with the following code:

```

SELECT /* TASK04 */ cust_city,sum(amount_sold)
 FROM sales s, customers c
 WHERE s.cust_id=c.cust_id and country_id = 52787
 GROUP BY cust_city
 ORDER BY 2
 /
 SELECT * FROM table(dbms_xplan.display_cursor)

```

Below the worksheet is a 'Script Output' window showing the results of the execution:

```

SQL_ID gx6h0s3w@uqwc, child number 0
SELECT /* TASK04 */ cust_city,sum(amount_sold) FROM sales s, customers
c WHERE s.cust_id=c.cust_id and country_id = 52787 GROUP BY cust_city
ORDER BY 2
Plan hash value: 2854028564

| Id | Operation | Name | Rows | Bytes | Cost (%CPU)| Time | Pstart| Pstop |
|---|---|---|---|---|---|---|---|---|
| 0 | SELECT STATEMENT | | | | 537 (100) | | | |
| 1 | SORT ORDER BY | | 71 | 2130 | 537 (2) | 00:00:01 | | |
| 2 | HASH GROUP BY | | 71 | 2130 | 537 (2) | 00:00:01 | | |
| * 3 | HASH JOIN | | 9762 | 285K | 537 (2) | 00:00:01 | | |
| 4 | TABLE ACCESS BY INDEX ROWID BATCHED | CUSTOMERS | 75 | 1500 | 18 (0) | 00:00:01 | | |
| * 5 | INDEX RANGE SCAN | CUST_COUNTRY_ID_IDX | 75 | | 1 (0) | 00:00:01 | | |
| 6 | PARTITION RANGE ALL | | 918K | 3973K | 517 (2) | 00:00:01 | 1 | 28 |
| 7 | TABLE ACCESS FULL | SALES | 918K | 3973K | 517 (2) | 00:00:01 | 1 | 28 |

Predicate Information (identified by operation id):

3 - access("S"."CUST_ID"="C"."CUST_ID")
5 - access("COUNTRY_ID"=52787)

Note

```

On the right side of the interface, there is a vertical sidebar with the text "Oracle University and Error. You are not a Valid Partner use only".

- d. Drop the set\_cursor\_sharing trigger by connecting as user sys. Use the w8\_s3\_d.sql file.

The screenshot shows the Oracle SQL Developer interface. In the top tab bar, three tabs are open: 'ws08.sql', 'w8\_s3\_c.sql', and 'w8\_s3\_d.sql'. The 'w8\_s3\_d.sql' tab is active, showing a SQL Worksheet with the following code:

```

DROP TRIGGER set_cursor_sharing;

```

Below the worksheet is a 'Script Output' window showing the results of the execution:

```

trigger SET_CURSOR_SHARING dropped.

```

4. Use a user-defined bind variable.

Assume that the identified SQL statements in ws08.sql can be modified. Use a user-defined bind variable.

- Connect to the sh schema in SQL Developer.
- Execute the following SQL statement and verify if a user-defined bind variable was used. You can use the w8\_s4\_b.sql file.

```
variable country_id number
exec :country_id:= 52790

SELECT /* ACS */ cust_city, sum (amount_sold)
FROM sales s, customers c
WHERE s.cust_id=c.cust_id and country_id =:country_id
GROUP BY cust_city
ORDER BY 2
/
SELECT * FROM table (dbms_xplan.display_cursor)
/
```

The screenshot shows the Oracle SQL Developer interface. In the top tab bar, the active tab is 'w8\_s4\_b.sql'. The main workspace contains the following SQL code:

```

variable country_id number
exec :country_id:= 52790

SELECT /* ACS */ cust_city, sum(amount_sold)
FROM sales s, customers c
WHERE s.cust_id=c.cust_id and country_id = :country_id
GROUP BY cust_city
ORDER BY 2
/
SELECT * FROM table(dbms_xplan.display_cursor)
/

```

Below the code, the 'Script Output' window displays the results of the execution:

```

SQL_ID 6agjrk4kckdk, child number 0

SELECT /* ACS */ cust_city, sum(amount_sold) FROM sales s, customers c
WHERE s.cust_id=c.cust_id and country_id = :country_id GROUP BY
cust_city ORDER BY 2

Plan hash value: 947950142

| Id | Operation | Name | Rows | Bytes | Cost (%CPU)| Time | Pstart| Pstop |
|---|---|---|---|---|---|---|---|---|
| 0 | SELECT STATEMENT | | | | 983 (100)| | | |
| 1 | SORT ORDER BY | | 620 | 18600 | 983 (6) | 00:00:01 | | |
| 2 | HASH GROUP BY | | 620 | 18600 | 983 (6) | 00:00:01 | | |
|* 3 | HASH JOIN | | 918K | 26M | 942 (2) | 00:00:01 | | |
|* 4 | TABLE ACCESS FULL | CUSTOMERS | 18520 | 361K | 423 (1) | 00:00:01 | | |
| 5 | PARTITION RANGE ALL| | 918K | 8973K | 517 (2) | 00:00:01 | | |
| 6 | TABLE ACCESS FULL | SALES | 918K | 8973K | 517 (2) | 00:00:01 | | |

Predicate Information (identified by operation id):

 3 - access("S"."CUST_ID"="C"."CUST_ID")
 4 - filter("COUNTRY_ID"=:COUNTRY_ID)
```

The execution completed in 0.585 seconds.

- c. Execute the following SQL statement and verify if a user-defined bind variable was used. You can use the w8\_s4\_c.sql file.

```

variable country_id number
exec :country_id:= 52787

SELECT /* ACS */ cust_city, sum (amount_sold)
FROM sales s, customers c
WHERE s.cust_id=c.cust_id and country_id =:country_id
GROUP BY cust_city
ORDER BY 2
/
SELECT * FROM table (dbms_xplan.display_cursor)
/
```

The screenshot shows the Oracle SQL Worksheet interface. In the top tab bar, the active tab is 'w8\_s4\_c.sql'. Below the tabs, there's a toolbar with icons for file operations and a status bar indicating '0.28799999 seconds'. The main area is divided into two panes. The left pane contains the following PL/SQL code:

```

variable_country_id number
exec :country_id := 52787

SELECT /* ACS */ cust_city,sum(amount_sold)
FROM sales s, customers c
WHERE s.cust_id=c.cust_id and country_id = :country_id
GROUP BY cust_city
ORDER BY 2
/
SELECT * FROM table(dbms_xplan.display_cursor)

```

The right pane shows the 'Script Output' results, which include the execution plan for the query. The plan hash value is 2854028564. The execution plan details the following steps:

| Id | Operation                           | Name                | Rows | Bytes | Cost (%CPU) | Time     | Pstart | Pstop |
|----|-------------------------------------|---------------------|------|-------|-------------|----------|--------|-------|
| 0  | SELECT STATEMENT                    |                     |      |       | 537 (100)   |          |        |       |
| 1  | SORT ORDER BY                       |                     | 71   | 2130  | 537 (2)     | 00:00:01 |        |       |
| 2  | HASH GROUP BY                       |                     | 71   | 2130  | 537 (2)     | 00:00:01 |        |       |
| *  | HASH JOIN                           |                     | 9762 | 285K  | 537 (2)     | 00:00:01 |        |       |
| 4  | TABLE ACCESS BY INDEX ROWID BATCHED | CUSTOMERS           | 75   | 1500  | 18 (0)      | 00:00:01 |        |       |
| *  | INDEX RANGE SCAN                    | CUST_COUNTRY_ID_IDX | 75   |       | 1 (0)       | 00:00:01 |        |       |
| 6  | PARTITION RANGE ALL                 |                     | 918K | 3973K | 517 (2)     | 00:00:01 | 1      | 28    |
| 7  | TABLE ACCESS FULL                   | SALES               | 918K | 3973K | 517 (2)     | 00:00:01 | 1      | 28    |

Below the execution plan, the 'Predicate Information (identified by operation id):' section shows the condition: 3 - access("S"."CUST\_ID"="C"."CUST\_ID")

- d. Even though the `country_id` user bind variable was defined, the optimizer created two plans. This is different optimizer behavior from Oracle 10g. Try to explain what made it possible.
  - 1) Do you know the Bind Variable Peeking feature?
  - 2) Do you know the Adaptive Cursor Sharing feature?
- 5. Using SQL Plan Baseline
  - a. Load the following two plans found in Step b and c into SQL Plan Baseline to maintain these plans over time. Use the following same code. You can use the `w8_s5_a.sql` and `w8_s5_a1.sql` files. **Note:** If you get an “Insufficient Privilege” error, grant the following permission to the `sh` user.

```
GRANT ADMINISTER SQL MANAGEMENT OBJECT TO SH;
```

```
var res number ;
```

```
exec :res := dbms_spm.load_plans_from_cursor_cache(sql_id =>
'<<Please enter the SQL_ID value>>', plan_hash_value => '<<Please enter
the PLAN_HASH_VALUE value>>')
/
```

- b. Execute the SQL statement and verify if two plans were successfully loaded into the SQL plan baseline. You can use the w8\_s5\_b.sql file.

```
select sql_text, sql_handle, plan_name, enabled, accepted
from dba_sql_plan_baselines
where sql_text like 'SELECT /* ACS %';
```

| SQL_TEXT                                                                                                                                     | SQL_HANDLE | PLAN_NAME | ENABLED | ACCEPTED |
|----------------------------------------------------------------------------------------------------------------------------------------------|------------|-----------|---------|----------|
| 1 SELECT /* ACS */ cust_city, sum(amount_sold)FROM sales s, customers c WHERE... SQL_b689bc3d113261fa SQL_PLAN_bd2dw7n8m4sgu9026c799 YES YES |            |           |         |          |
| 2 SELECT /* ACS */ cust_city, sum(amount_sold)FROM sales s, customers c WHERE... SQL_b689bc3d113261fa SQL_PLAN_bd2dw7n8m4sgu8b583622 YES YES |            |           |         |          |

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Error : You are not a Valid Partner use only

## **Workshop 9**

### **Chapter 22**

## Workshop 9: Overview

---

### Workshop Overview

In this workshop, you will perform the following task:

Use SQL Plan Baseline to associate a hinted execution plan with a hard-coded SQL statement.

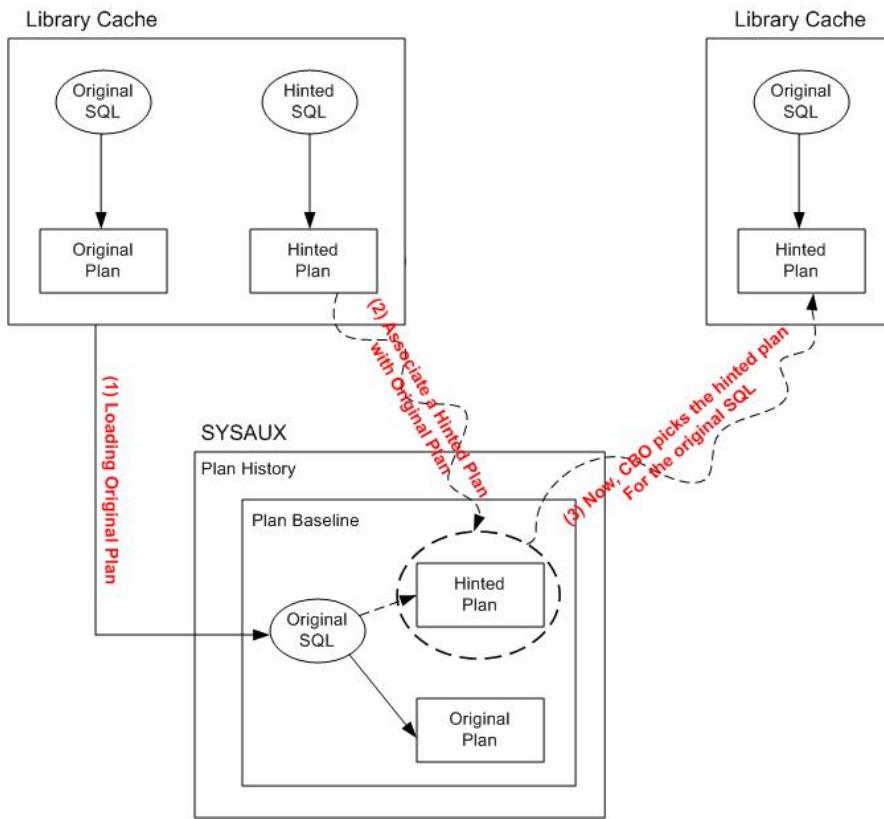
### Scripts Used in the Workshop

- `setup09.sql`
- `ws09.sql`
- `cleanp09.sql`
- `w9_s2_a.sql`
- `w9_s2_b.sql`
- `w9_s2_c.sql`
- `w9_s2_d.sql`
- `w9_s3_a.sql`
- `w9_s3_b.sql`
- `w9_s4_a.sql`
- `w9_s4_b.sql`
- `w9_s5_a.sql`
- `w9_s4_c.sql`
- `w9_s4_c1.sql`
- `w9_s5_b.sql`
- `w9_s5_b1.sql`
- `w9_s5_b2.sql`

## Workshop 9: Using SQL Plan Baseline to Manage a Better Execution Plan

### Overview

Oracle has introduced several methods to maintain the stable execution plans of SQL statements such as hints, stored outlines, and SQL profile. In Oracle Database, you can manage the execution plan through a methodology called SQL Plan Baseline. This feature allows the optimizer to make the right decision choosing a verified or tested execution plan in SQL Plan Baseline even if the optimizer generates a new plan during hard parse. In this workshop, you learn how to use this feature to associate a hinted execution plan with a hard-coded SQL statement.



### Tasks

1. Initial setup
    - a. Connect to sh schema in SQL Developer.
    - b. Execute the `setup09.sql` script to set up the environment for this workshop.
- Note:** The scripts used in this workshop are available in the workshops folder located in `/home/oracle/labs/workshops`.

```
DROP TABLE sh.cust
```

```

/
CREATE TABLE sh.cust AS SELECT * FROM sh.customers
/
DROP INDEX sh.cust_yob_bix
/
CREATE BITMAP INDEX sh.cust_yob_bix
ON sh.cust (cust_year_of_birth)
NOLOGGING
COMPUTE STATISTICS
/
DROP INDEX sh.cust_gender_bix
/
CREATE BITMAP INDEX sh.cust_gender_bix
ON sh.cust (cust_gender)
NOLOGGING
COMPUTE STATISTICS
/
exec DBMS_STATS.GATHER_TABLE_STATS('SH','CUST')
/

```

- c. Review and execute the SQL statement in ws09.sql.

**Note:** Assume that you have identified this hard-coded SQL statement. This has to be tuned, but there are only a few options available to improve performance.

**--Original SQL:**

```

SELECT /*+ INDEX_COMBINE (cust cust_gender_bix, cust_yob_bix) */
*
FROM sh.cust
WHERE cust_year_of_birth < 70
AND cust_gender = 'M'
/

```

2. Review the identified SQL statement.

- a. Execute the SQL statement in ws09.sql. And check sql\_id and plan\_hash\_value using v\$sqlarea. Use the following sample code. You can use also w9\_s2\_a.sql file.

**Note:** The workshop is used to see SQL plan management. The query in ws09.sql does not return any value on purpose. It is only to see which plan can be used. And the original query will use the hinted execution plan, not the original plan. That is the whole purpose of workshop 9.

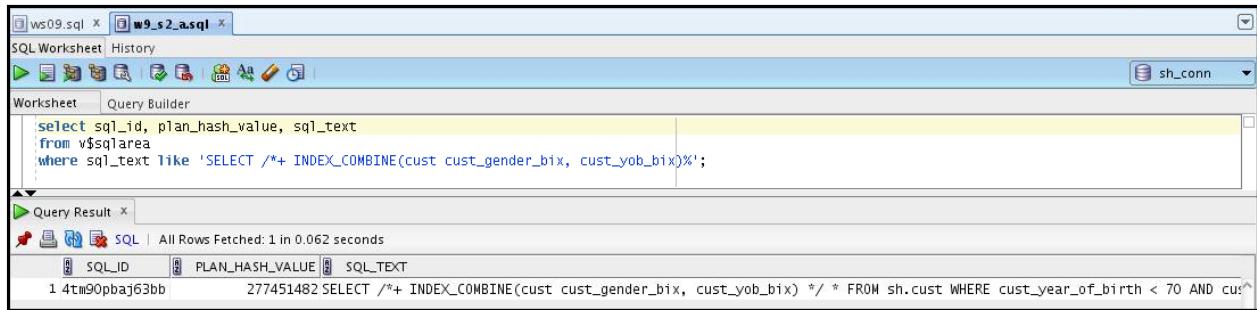
In case SQL cannot be modified in the real world and you know the better plan, you can associate the original SQL with the new plan (hinted plan).

```

select sql_id, plan_hash_value, sql_text
from v$sqlarea

```

```
where sql_text like 'SELECT /*+ INDEX_COMBINE(cust
cust_gender_bix, cust_yob_bix)%';
```



The screenshot shows the Oracle SQL Worksheet interface. In the top tab bar, there are two tabs: 'ws09.sql' and 'w9\_s2\_a.sql'. The main area is titled 'Worksheet' and contains a SQL query:`select sql_id, plan_hash_value, sql_text
from v$sqlarea
where sql_text like 'SELECT /*+ INDEX_COMBINE(cust cust_gender_bix, cust_yob_bix)%';`

Below the query, the 'Query Result' tab is selected, showing the output:

| SQL_ID        | PLAN_HASH_VALUE | SQL_TEXT                                                                                                            |
|---------------|-----------------|---------------------------------------------------------------------------------------------------------------------|
| 4tm90pbaj63bb | 277451482       | SELECT /*+ INDEX_COMBINE(cust cust_gender_bix, cust_yob_bix) */ * FROM sh.cust WHERE cust_year_of_birth < 70 AND cu |

The status bar at the bottom indicates 'All Rows Fetched: 1 in 0.062 seconds'.

- b. Execute the following sample code and review the execution plan of the SQL statement. You can use the w9\_s2\_b.sql file.

```
SELECT * FROM table (dbms_xplan.display_cursor ('<<Please Enter the
SQL_ID value>>')) ;
```

The screenshot shows the Oracle SQL Developer interface. In the top tab bar, there are two tabs: 'ws09.sql' and 'w9\_s2\_b.sql'. The 'w9\_s2\_b.sql' tab is active. Below the tabs, the title bar says 'SQL Worksheet History'. The main area is divided into two panes: 'Worksheet' and 'Query Builder'. The 'Worksheet' pane contains the SQL query:

```
SELECT * FROM table (dbms_xplan.display_cursor('4tm90pbaj63bb'));
```

The 'Query Result' pane displays the execution plan. It starts with the PLAN\_TABLE\_OUTPUT section:

```
PLAN_TABLE_OUTPUT
1 SQL_ID 4tm90pbaj63bb, child number 0
2 -----
3 SELECT /*+ INDEX_COMBINE(cust cust_gender_bix, cust_yob_bix) */ * FROM
4 sh.cust WHERE cust_year_of_birth < 70 AND cust_gender = 'M'
5
6 Plan hash value: 277451482
7
8 -----
9 | Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time
10 -----
11 | 0 | SELECT STATEMENT | | | | 4 (100) |
12 | 1 | TABLE ACCESS BY INDEX ROWID BATCHED | CUST | 1 | 189 | 4 (0) 00:00:01
13 | 2 | BITMAP CONVERSION TO ROWIDS | | | |
14 | 3 | BITMAP AND | | | |
15 | 4 | BITMAP MERGE | | | |
16 |/* 5 | BITMAP INDEX RANGE SCAN | CUST_YOB_BIX | | |
17 |/* 6 | BITMAP INDEX SINGLE VALUE | CUST_GENDER_BIX | | |
18 -----
19
20 Predicate Information (identified by operation id):
21 -----
22
23 5 - access("CUST_YEAR_OF_BIRTH"<70)
24 filter("CUST_YEAR_OF_BIRTH"<70)
25 6 - access("CUST_GENDER='M'")
26
```

Below the execution plan, it says 'Predicate Information (identified by operation id):' followed by the predicates for operations 5 and 6.

**Note:** The BITMAP MERGE operation is used. Total Cost is 4.

- c. Load the execution plan of the SQL statement in ws09.sql into the SQL Plan Baseline. You can use the w9\_s2\_c.sql file.

```
var res number ;
exec :res: = dbms_spm.load_plans_from_cursor_cache (sql_id =>
'<<Please enter the original SQL_ID value>>', plan_hash_value =>
'<<Please enter the &original_plan_hash_value>>');
```

The screenshot shows the Oracle SQL Worksheet interface. In the top tab bar, there are three tabs: 'ws09.sql', 'w9\_s2\_b.sql', and 'w9\_s2\_c.sql'. The 'w9\_s2\_c.sql' tab is active. The main workspace contains the following PL/SQL code:

```

var_res number ;
exec :res := dbms_spml.load_plans_from_cursor_cache(sql_id=>'4tm90pbaj63bb', plan_hash_value=>'277451482');

```

Below the workspace is a 'Script Output' window with the message: 'Task completed in 0.295 seconds'. At the bottom of the window, it says 'anonymous block completed'.

- d. Check the loaded plan in SQL Plan Baseline by using the following sample code, or you can use w9\_s2\_d.sql file.

```

select sql_handle, plan_name, sql_text, enabled, accepted
from dba_sql_plan_baselines
where sql_text like '%SELECT /*+ INDEX_COMBINE (cust
cust_gender_bix, cust_yob_bix)%';

```

The screenshot shows the Oracle SQL Worksheet interface. The 'w9\_s2\_c.sql' tab is active. The workspace contains the same query as above:

```

select sql_text, sql_handle, plan_name, enabled, accepted
from dba_sql_plan_baselines
where sql_text like 'SELECT /*+ INDEX_COMBINE(cust cust_gender_bix, cust_yob_bix)%';

```

Below the workspace is a 'Query Result' window. It shows one row of data with the following columns:

| SQL_TEXT                                                                                                                                     | SQL_HANDLE | PLAN_NAME | ENABLED | ACCEPT |
|----------------------------------------------------------------------------------------------------------------------------------------------|------------|-----------|---------|--------|
| 1 SELECT /*+ INDEX_COMBINE(cust cust_gender_bix, cust_yob_bix) */ /*FROM sh.c... SQL_db38613f52934cc0 SQL_PLAN_buf317x996m60f6d8b2e3 YES YES |            |           |         |        |

3. Find a better execution plan.

- a. Execute the following hinted SQL statement and check `sql_id` and `plan_hash_value`. Use the w9\_s3\_a.sql file.

```

SELECT /*+INDEX(cust cust_yob_bix) */ *
FROM sh.cust
WHERE cust_year_of_birth < 70
AND cust_gender = 'M'
/
select sql_id, plan_hash_value, sql_text
from v$sqlarea
where sql_text like 'SELECT /*+INDEX(cust cust_yob_bix)%';

```

The screenshot shows the Oracle SQL Worksheet interface. In the top tab bar, there are three tabs: ws09.sql, w9\_s3\_a.sql (which is the active tab), and w9\_s2\_a.sql. The title bar indicates "SQL Worksheet History" and "sh\_conn". The main area is titled "Worksheet" and contains the following SQL code:

```
SELECT /*+INDEX(cust cust_yob_bix) */ *
FROM sh.cust
WHERE cust_year_of_birth < 70
AND cust_gender = 'M'
/
select sql_id, plan_hash_value, sql_text
from v$sqlarea
where sql_text like 'SELECT /*+INDEX(cust cust_yob_bix)%';
```

Below the code, the "Script Output" section shows the results of the query:

```
no rows selected
```

| SQL_ID        | PLAN_HASH_VALUE | SQL_TEXT                                                                                                 |
|---------------|-----------------|----------------------------------------------------------------------------------------------------------|
| Onvxdkfrhpny5 | 1264022301      | SELECT /*+INDEX(cust cust_yob_bix) */ * FROM sh.cust WHERE cust_year_of_birth < 70 AND cust_gender = 'M' |

A message at the bottom of the output pane says "Task completed in 2.831 seconds".

- b. Execute the following sample code and review the execution plan of the SQL statement. You can use the w9\_s3\_b.sql file.

```
SELECT * FROM table (dbms_xplan.display_cursor ('<<Please enter the
SQL_ID value>>'));
```

```

SELECT * FROM table(dbms_xplan.display_cursor('Onvxdkfrhpny5'));

```

PLAN\_TABLE\_OUTPUT

```

1 SQL_ID Onvxdkfrhpny5, child number 0
2 -
3 SELECT /*+INDEX(cust cust_yob_bix) */ * FROM sh.cust WHERE
4 cust_year_of_birth < 70 AND cust_gender = 'M'
5
6 Plan hash value: 1264022301
7
8
9 | Id | Operation
10 | | Name
11 | 0 | SELECT STATEMENT
12 |/* 1 | TABLE ACCESS BY INDEX ROWID BATCHED| CUST
13 | 2 | BITMAP CONVERSION TO ROWIDS
14 |/* 3 | BITMAP INDEX RANGE SCAN | CUST_YOB_BIX
15 -
16
17 Predicate Information (identified by operation id):
18 -
19
20 1 - filter("CUST_GENDER"='M')
21 3 - access("CUST_YEAR_OF_BIRTH"<70)
22 filter("CUST_YEAR_OF_BIRTH"<70)
23

```

**Note:** The BITMAP MERGE is avoided. Total cost is 2.

4. Load a hinted execution plan into SQL Plan Baseline.
  - a. Associate the better execution plan generated in step b of Task 3 with the original SQL statement. Use the w9\_s4\_a.sql file.

```

var res number;
exe :res := dbms_spm.load_plans_from_cursor_cache(sql_id =>
'<<Please enter the &hinted_SQL_ID value>>', plan_hash_value
=>'<<Please enter the &hinted_PLAN_HASH_VALUE>>', sql_handle =>
'<<Please enter the '&SQL_HANDLE_for_original>>');

```

The screenshot shows the Oracle SQL Worksheet interface. In the 'Worksheet' tab, there is a code block containing a PL/SQL anonymous block:

```

var res number;
exec :res := dbms_spm.load_plans_from_cursor_cache(sql_id=> '0nvxdkfrhpny5', plan_hash_value=> '1264022301' , sql_handle=> 'SQL_bd38613f52934cc0');

```

The output pane below shows the message "Task completed in 1.153 seconds" and "anonymous block completed". The status bar at the bottom indicates "1.153 seconds".

- b. Execute the following sample code and verify that the better plan is loaded successfully in SQL Plan Baseline for the original SQL statement. You can use the w9\_s4\_b.sql file.

```

select sql_text, sql_handle, plan_name, enabled, accepted
from dba_sql_plan_baselines
where sql_text like 'SELECT /*+ INDEX_COMBINE(cust%';

```

The screenshot shows the Oracle SQL Worksheet interface. A query has been run:

```

select sql_text, sql_handle, plan_name, enabled, accepted
from dba_sql_plan_baselines
where sql_text like 'SELECT /*+ INDEX_COMBINE(cust%';

```

The results are displayed in a table titled "Query Result". The table has columns: SQL\_TEXT, SQL\_HANDLE, PLAN\_NAME, ENABLED, and ACCEPTED. There are two rows of data:

| SQL_TEXT                                                                                                                                | SQL_HANDLE           | PLAN_NAME                      | ENABLED | ACCEPTED |
|-----------------------------------------------------------------------------------------------------------------------------------------|----------------------|--------------------------------|---------|----------|
| 1 SELECT /*+ INDEX_COMBINE(cust cust_gender_bix, cust_yob_bix) /* *FROM sh.c... SQL_bd38613f52934cc0 SQL_PLAN_buf317x996#60f6d8b2e3 YES | SQL_bd38613f52934cc0 | SQL_PLAN_buf317x996#60f6d8b2e3 | YES     | YES      |
| 2 SELECT /*+ INDEX_COMBINE(cust cust_gender_bix, cust_yob_bix) /* *FROM sh.c... SQL_bd38613f52934cc0 SQL_PLAN_buf317x996#60cdc46b71 YES | SQL_bd38613f52934cc0 | SQL_PLAN_buf317x996#60cdc46b71 | YES     | YES      |

The status bar at the bottom indicates "All Rows Fetched: 2 in 0.012 seconds".

**Note:** The second plan is the hinted plan loaded.

- c. If the original plan is not needed, drop the original plan (**option1**) from SQL Plan Baseline or fix the better plan to guarantee the plan for the next hard parse (**option2**). Use the w9\_s4\_c.sql (**option1**) file to drop the original plan file and then execute the w9\_s4\_c1.sql file to check whether the original plan is dropped or not.

#### --Option 1

```

var res number;
exec :res :=DBMS_SPM.DROP_SQL_PLAN_BASELINE ('<< Please enter the
&original_SQL_HANDLE value>>', '<<Please enter the
&original_PLAN_NAME value>>');

```

#### --Option 2

```

var cnt number
exec :cnt := dbms_spm.alter_sql_plan_baseline(sql_handle =>'<<
Please enter the SQL_HANDLE value>>', plan_name=> '<<Please enter the
PLAN_NAME value>>');

```

SQL Worksheet History  
sh\_conn  
Worksheet Query Builder

```
var res number;
exec :res :=DBMS_SPM.DROP_SQL_PLAN_BASELINE ('SQL_bd38613f52934cc0','SQL_PLAN_buf317x996m60f6d8b2e3');
```

Script Output X  
Task completed in 1.53 seconds  
anonymous block completed

```
select sql_text, sql_handle, plan_name, enabled, accepted
from dba_sql_plan_baselines
where sql_text like 'SELECT /*+ INDEX_COMBINE(cust%';
```

SQL Worksheet History  
sh\_conn  
Worksheet Query Builder

```
select sql_text, sql_handle, plan_name, enabled, accepted
from dba_sql_plan_baselines
where sql_text like 'SELECT /*+ INDEX_COMBINE(cust%';
```

Query Result X  
All Rows Fetched: 1 in 0.013 seconds

| SQL_TEXT                                                                                                                                 | SQL_HANDLE | PLAN_NAME | ENABLED | ACCEPTED |
|------------------------------------------------------------------------------------------------------------------------------------------|------------|-----------|---------|----------|
| 1 SELECT /*+ INDEX_COMBINE(cust cust_gender_bix, cust_yob_bix) */ /*FROM sh.c... SQL_bd38613f52934cc0 SQL_PLAN_buf317x996m60cdc46b71 YES |            |           |         | YES      |

## 5. Test the SQL statement with the loaded plan.

- Clean up the shared pool for testing in the next step. You can use the `w9_s5_a.sql` file. Use `sys` user to execute this statement.

```
ALTER SYSTEM FLUSH SHARED_POOL;
```

0.33399999 second  
Worksheet Query Builder

```
ALTER SYSTEM FLUSH SHARED_POOL;
```

Script Output X  
Task completed in 0.334 seconds  
system FLUSH altered.

- Execute the SQL statements (**option 1, option 2, and option 3**) and check if the optimizer uses the better execution plan for the original SQL statement. Use the `w9_s5_b.sql`, `w9_s5_b1.sql`, and then `w9_s5_b2.sql` files.

### --Option 1:

```
EXPLAIN PLAN FOR
```

```

SELECT /*+ INDEX_COMBINE(cust cust_gender_bix, cust_yob_bix) */
*
FROM sh.cust
WHERE cust_year_of_birth < 70
AND cust_gender = 'M'
/
SELECT * FROM table (DBMS_XPLAN.DISPLAY);

```

The screenshot shows the Oracle SQL Worksheet interface. The top window is titled "Worksheet" and contains the following SQL code:

```

EXPLAIN PLAN FOR SELECT /*+ INDEX_COMBINE(cust cust_gender_bix, cust_yob_bix) */ *
 FROM sh.cust
 WHERE cust_year_of_birth < 70
 AND cust_gender = 'M'
/
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

```

The bottom window is titled "Script Output" and displays the execution results:

```

plan FOR succeeded.
PLAN_TABLE_OUTPUT

Plan hash value: 1264022301

| Id | Operation | Name | Rows | Bytes | Cost (%CPU)| Time |
| 0 | SELECT STATEMENT | | 1 | 189 | 2 (0) | 00:00:01 |
|* 1 | TABLE ACCESS BY INDEX ROWID BATCHED | CUST | 1 | 189 | 2 (0) | 00:00:01 |
|* 2 | BITMAP CONVERSION TO ROWIDS | | | | | |
|* 3 | BITMAP INDEX RANGE SCAN | CUST_YOB_BIX | | | | |

Predicate Information (identified by operation id):

1 - filter("CUST_GENDER"='M')
3 - access("CUST_YEAR_OF_BIRTH"<70)
 filter("CUST_YEAR_OF_BIRTH"<70)

Note

- SQL plan baseline "SQL_PLAN_buf317x996m60cdc46b71" used for this statement
21 rows selected

```

### --Option 2:

```

SELECT /*+ INDEX_COMBINE(cust cust_gender_bix, cust_yob_bix) */
*
FROM sh.cust
WHERE cust_year_of_birth < 70
AND cust_gender = 'M'
/

```

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

```
select sql_id, sql_plan_baseline from v$sqlarea where sql_text
like 'SELECT /*+ INDEX_COMBINE(cust%');
```

The screenshot shows the Oracle SQL Worksheet interface. The top menu bar includes 'SQL Worksheet' and 'History'. Below the menu is a toolbar with various icons. A connection named 'sh\_conn' is selected. The main area has two tabs: 'Worksheet' and 'Query Builder', with 'Worksheet' currently active. The worksheet contains the following SQL code:

```
SELECT /*+ INDEX_COMBINE(cust cust_gender_bix, cust_yob_bix) */ *
 FROM sh.cust
 WHERE cust_year_of_birth < 70
 AND cust_gender = 'M'
/
SELECT sql_id, sql_plan_baseline from v$sqlarea where sql_text like 'SELECT /*+ INDEX_COMBINE(cust%');
```

Below the code, there is a 'Query Result' window. It shows a single row of data:

| SQL_ID          | SQL_PLAN_BASELINE              |
|-----------------|--------------------------------|
| 1 4tm90pbaj63bb | SQL_PLAN_buf317x996m60cdc46b71 |

The result set is described as 'All Rows Fetched: 1 in 0.006 seconds'.

**--Option 3:**

```
SET LINESIZE 150
SET PAGESIZE 2000
SELECT t.* FROM TABLE (DBMS_XPLAN.DISPLAY_SQL_PLAN_BASELINE ('<<
Please enter SQL_HANDLE value>>')) t;
```

The screenshot shows the Oracle SQL Worksheet interface. In the top pane, a query is run:

```
SET LINESIZE 150
SET PAGESIZE 2000
SELECT t.* FROM TABLE(DBMS_XPLAN.DISPLAY_SQL_PLAN_BASELINE('SQL_bd38613f52934cc0')) t;
```

The bottom pane displays the results of the query, titled "PLAN\_TABLE\_OUTPUT".

```
SQL handle: SQL_bd38613f52934cc0
SQL text: SELECT /*+ INDEX_COMBINE(cust cust_gender_bix, cust_yob_bix) */ * FROM sh.cust WHERE cust_year_of_birth < 70 AND cust_gender = 'M'

Plan name: SQL_PLAN_buf317x996m60cdc46b71 Plan id: 3452201841
Enabled: YES Fixed: NO Accepted: YES Origin: MANUAL-LOAD
Plan rows: From dictionary

Plan hash value: 1264022301

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time |
|---|---|---|---|---|---|---|
| 0 | SELECT STATEMENT | | | | 2 (100) | |
|* 1 | TABLE ACCESS BY INDEX ROWID BATCHED | CUST | 1 | 189 | 2 (0) | 00:00:01 |
| 2 | BITMAP CONVERSION TO ROWIDS | | | | 0 (0) | |
|* 3 | BITMAP INDEX RANGE SCAN | CUST_YOB_BIX | | | 0 (0) | |

Predicate Information (identified by operation id):
1 - filter("CUST_GENDER"='M')
3 - access("CUST_YEAR_OF_BIRTH"><70)
 filter("CUST_YEAR_OF_BIRTH"><70)
```

## 6. Clean up the environment.

Execute the `cleanup09.sql` script.

```
select sql_text, sql_handle, plan_name, enabled, accepted
from dba_sql_plan_baselines
where sql_text like 'SELECT /*+ INDEX_COMBINE(cust%'
/
var res number;
exec :res :=DBMS_SPM.DROP_SQL_PLAN_BASELINE ('<<Please enter the
SQL_HANDLE value>>');
/
DROP TABLE sh.cust
/
```