Oracle Database 19c: Tuning

Table of Contents

Practices for Lesson 1: Course Introduction 5
Practices for Lesson 1 6
Practices for Lesson 2: Introduction to SQL Tuning 7
Practices for Lesson 2: Overview 8
Practice 2-1: Using SQL Developer 9
Practices for Lesson 3: Using Application Tracing Tools 21
Practices for Lesson 3: Overview 22
Practice 3-1: Tracing Applications 23
Practices for Lesson 4: Optimizer Fundamentals 59
Practices for Lesson 4: Overview 60
Practice 4-1: Understanding Optimizer Decisions (Optional) 61
Practices for Lesson 5: Generating and Displaying Execution Plans 101
Practices for Lesson 5: Overview 102
Practice 5-1: Extracting an Execution Plan by Using SQL Developer 103
Practice 5-2: Extracting Execution Plans 108
Practices for Lesson 6: Interpreting Execution Plans and Enhancements 123
Practices for Lesson 6: Overview 124
Practice 6-1: Using Dynamic Plans 125
Practices for Lesson 7: Optimizer: Table and Index Access Paths 131
Practices for Lesson 7: Overview 132
Practice 7-1: Using Different Access Paths 133
Practices for Lesson 8: Optimizer: Join Operators 175
Practices for Lesson 8: Overview 176
Practice 8: Using Join Paths 177
Practices for Lesson 9: Other Optimizer Operators 189
Practices for Lesson 9: Overview 190
Practice 9-1: Using the Result Cache 191
Practice 9-2: Using Other Access Paths (Optional) 210
Practices for Lesson 10: Introduction to Optimizer Statistics Concepts 231
Practices for Lesson 10: Overview 232
Practice 10-1: Index Clustering Factor 233
Practice 10-2: Creating Expression Statistics 242
Practice 10-3: Enabling Automatic Statistics Gathering (Optional) 247
Practice 10-4: Using System Statistics (Optional) 269
Practices for Lesson 11: Using Bind Variables 279

Practices for Lesson 11: Overview 280
Practice 11-1: Using Adaptive Cursor Sharing 281
Practice 11-2: Using `CURSOR_SHARING` (Optional) 300

Practices for Lesson 12: SQL Plan Management 309
Practices for Lesson 12: Overview 310
Practice 12-1: Using SQL Plan Management (SPM) 311

Workshop 1 347
Workshop 1: Overview 348
Workshop 1: Enhancing the Performance of a SQL Query Statement 349

Workshop 2 357
Workshop 2: Overview 358
Workshop 2: Reviewing the Execution Steps of the SQL Statement 359

Workshop 3 365
Workshop 3: Overview 366
Workshop 3: Learn to Tune Sort Operation Using an Index in the `ORDER BY` Clauses 367

Workshop 4 379
Workshop 4: Overview 380
Workshop 4: Identifying and Tuning a Poorly Written SQL Statement 381

Workshop 5 387
Workshop 5: Overview 388
Workshop 5: Effects of Changing the Column Order in a Composite Index 389

Workshop 6 403
Workshop 6: Overview 404
Workshop-6: Using Information in the 10053 File to Tune a SQL Statement 405

Workshop 7 409
Workshop 7: Overview 410
Workshop 7: Understanding the Optimizer's Decision 411

Workshop 8 417
Workshop 8: Overview 418
Workshop 8: Tuning Strategy 419

Workshop 9 435
Workshop 9: Overview 436
Workshop 9: Using SQL Plan Baseline to Manage a Better Execution Plan 437

Practices for Lesson 1: Course Introduction

Practices for Lesson 1

There are no practices for lesson 1.

Practices for Lesson 2: Introduction to SQL Tuning

Practices for Lesson 2: Overview

Overview

This practice covers using SQL Developer.

Practice 2-1: Using SQL Developer

Overview

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

Tasks

1. Start Database. To start up a database from the command line, use SQL*Plus to connect to Oracle with administrator privileges and then issue the STARTUP command.
   a) Open terminal from the desktop and switch to oracle user:
      `su - oracle`
   b) Enter `sqlplus / as sysdba`
   c) Start the database using the command `startup`. Enter `startup`
   d) Enter following commands to open the current PDB
      `Alter pluggable database all open;`
   e) Exit SQL*Plus command-line. Enter `EXIT`.
   f) Start the listener service by typing the following command:
      `lsnrctl START`

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

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

a. Connection Name	`sh_connection`
b. Username	`sh`
c. Password	`sh`
d. Hostname	`localhost`
e. Port	`1521`
f. Service name	`orclpdb`

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:

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

5. 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.

6. 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.

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

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:

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:

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.

    a. 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;
```

    b. Execute by pressing Ctrl + Enter or by clicking the Run Statement icon .

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COI
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	MK_MAN	13000	
15	205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-02	AC_MGR	12008	

c. Execute by pressing the F5 key or by clicking the Execute Script icon .

11. In the same SQL Worksheet, create and execute a simple anonymous block that outputs "Hello World."

    a. Enable `SET SERVEROUTPUT ON` to display the output of the `DBMS_OUTPUT` package statements. Select **View > Dbms Output.**

    b. 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:

Practices for Lesson 3: Using Application Tracing Tools

Practices for Lesson 3: Overview

Overview

This practice covers the following topics:

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

Practice 3-1: Tracing Applications

Lab Solution

Copy lab files in /home/oracle directory using following commands: (Run as root)

```
ln -s /headless/Desktop/oracle19c-labs/tuning/labs/ /home/oracle/
chmod -R 777 /headless/Desktop/oracle19c-labs/tuning/labs
```

Overview

In this practice, you define a service and use it to generate traces. You then interpret the 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:

Tasks

1. Your environment should be initialized with the `at_setup.sql` script as shown 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;
alter session set container = orclpdb;

drop tablespace tracetbs including contents and datafiles;

drop tablespace tracetbs3 including contents and datafiles;

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

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

connect trace/trace@orclpdb
```

```
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;
```

Execute the `at_setup.sql`, to prepare for this lab activity

a. Open terminal
b. Navigate to Application_Tracing folder usning the command `cd /home/oracle/labs/solutions/Application_Tracing`
c. Connect to your database instance as the SYS
   `sqlplus / as sysdba`
d. Execute the `at_setup.sql` script as follows
   `SQL>` @at_setup.sql

2. You now execute seven workload scripts that are traced. All workload scripts run under the `ORCLPDB` 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.

```
$ cd /home/oracle/labs/solutions/Application_Tracing
$ ./run_tracep0.sh


…
Connected to:…

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

Session altered.
```

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

...

$
```

3. Enable tracing for ORCLPDB service.

```
oracle@edvmr1p0 Application_Tracing]$ sqlplus / as sysdba

SQL*Plus: Release 19.0.0.0.0 - Production on Wed Jun 3 07:17:15 2020
Version 19.3.0.0.0

Copyright (c) 1982, 2019, Oracle.  All rights reserved.


Connected to:
Oracle Database 19c Enterprise Edition Release 19.0.0.0.0 -
Production
Version 19.3.0.0.0

SQL> select name from v$active_services;

NAME
----------------------------------------------------------------------------------
orclXDB
orcl
SYS$BACKGROUND
SYS$USERS
orclpdb

SQL> exec
dbms_monitor.serv_mod_act_trace_enable(service_name=>'orclpdb');

PL/SQL procedure successfully completed.

SQL> exit
```

4. When tracing for `ORCLPDB` is enabled and `run_tracep0.sh` has finished, execute the `run_tracep1.sh` script from your terminal session.

```
$ ./run_tracep1.sh
…

SQL>
SQL> connect trace/trace@orclpdb
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> EXEC DBMS_MONITOR.session_trace_enable;
PL/SQL procedure successfully completed.

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: 00:00:44.35
SQL>
```

```
SQL>
SQL>
SQL> connect trace/trace@orclpdb
Connected.
SQL>
SQL> alter session set tracefile_identifier='mytraceS1';

Session altered.

Elapsed: 00:00:00.00
SQL> EXEC DBMS_MONITOR.session_trace_enable;
PL/SQL procedure successfully completed.
Elapsed: 00:00:00.01

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: 00:00:13.72
SQL>
SQL> exit;

…

$
```

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

```
$ ./run_tracep2.sh
…

SQL> alter session set container = orclpdb;
Session altered.
SQL>
SQL> connect trace/trace@orclpdb
Connected.
SQL>
SQL> alter session set tracefile_identifier='mytraceP2';

Session altered.
SQL> EXEC DBMS_MONITOR.session_trace_enable;
PL/SQL procedure successfully completed.
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: 00:02:24.36
SQL>
SQL>
SQL>
SQL> connect trace/trace@orclpdb
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: 00:00:00.86
SQL>
SQL> exit; Disconnected …

$
```

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

```
$ ./run_tracep3.sh
…

SQL>
SQL> connect trace/trace@orclpdb
Connected.
SQL>
SQL> alter session set tracefile_identifier='mytraceP3';

Session altered.

SQL> EXEC DBMS_MONITOR.session_trace_enable;
PL/SQL procedure successfully completed.



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@orclpdb
Connected.
SQL>
SQL> create index sales_prod_cust_indx on sales(prod_id,cust_id);

Index created.

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

Session altered.

SQL> EXEC DBMS_MONITOR.session_trace_enable;
PL/SQL procedure successfully completed.
```

```
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 …
$
```

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

```
$ ./run_tracep4.sh
…

SQL>
SQL> connect trace/trace@orclpdb
Connected.
SQL>
SQL> alter session set tracefile_identifier='mytraceP4';

Session altered.
SQL> EXEC DBMS_MONITOR.session_trace_enable;
PL/SQL procedure successfully completed.

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: 00:00:21.27
SQL>
SQL>
SQL> connect trace/trace@orclpdb
Connected.
SQL>
SQL> alter session set tracefile_identifier='mytraceS4';

Session altered.

Elapsed: 00:00:00.01
SQL> EXEC DBMS_MONITOR.session_trace_enable;
PL/SQL procedure successfully completed.
Elapsed: 00:00:00.01

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: 00:00:00.87
SQL>
SQL>
SQL> exit;
Disconnected …
$
```

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

```
$ ./run_tracep5.sh
…

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

Session altered.
SQL>
```

```
SQL> EXEX DBMS_MONITOR.session_trace_enable;
PL/SQL procedure successfully completed.

SQL> delete from sales2;
0 rows deleted;

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@orclpdb
Connected.
SQL>
SQL> alter session set tracefile_identifier='mytraceS5';

Session altered.
SQL> EXEX DBMS_MONITOR.session_trace_enable;
PL/SQL procedure successfully completed.

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 …
$
```

9. 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@orclpdb
Connected.
SQL>
SQL> alter session set tracefile_identifier='mytraceP6';

Session altered.
SQL> EXEX DBMS_MONITOR.session_trace_enable;
PL/SQL procedure successfully completed.
SQL>
SQL> exec dbms_lock.sleep(30);
Connected.
SQL> update sales set amount_sold=amount_sold+1;

918843 rows updated.

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

SQL> exit;
Disconnected from Oracle Database 12c Enterprise Edition Release
12.2.0.1.0 - 64bit Production
[oracle@EDT3R13P1 Application_Tracing]$
PL/SQL procedure successfully completed.

SQL>
SQL> rollback;

Rollback complete.

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

$
```

Note: `run_tracep6.sh` takes a while to execute completely. Wait for some time to finish execution and then continue.

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

```
$ ./run_tracep7.sh
…
SQL>
SQL> connect trace/trace@orclpdb
Connected.
SQL> drop index SALES_CUST_INDX;
Index dropped.
SQL> alter session set tracefile_identifier='mytraceP7';

Session altered.

SQL> EXEC dbms_monitor.session_trace_enable;
PL/SQL procedure successfully completed.

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@orclpdb
Connected.
SQL>
SQL> create index sales_cust_indx on sales(cust_id);

Index created.

SQL>
```

```
SQL> exit;
Disconnected …
$
```

11. Disable tracing for your database.

```
[oracle@edvmr1p0 Application_Tracing]$ sqlplus / as sysdba

SQL*Plus: Release 19.0.0.0.0 - Production on Wed Jun 3 07:38:33 2020
Version 19.3.0.0.0

Copyright (c) 1982, 2019, Oracle.  All rights reserved.

Connected to:
Oracle Database 19c Enterprise Edition Release 19.0.0.0.0 -
Production
Version 19.3.0.0.0

SQL> exec
dbms_monitor.serv_mod_act_trace_disable(service_name=>'orclpdb');

PL/SQL procedure successfully completed.

SQL> exit
```

12. 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
orcl_ora_19313_mytraceP1.trc
orcl _ora_19313_mytraceP1.trm
orcl _ora_19355_mytraceS1.trc
orcl _ora_19355_mytraceS1.trm
orcl _ora_19382_mytraceP2.trc
orcl _ora_19382_mytraceP2.trm
orcl _ora_19467_mytraceS2.trc
orcl _ora_19467_mytraceS2.trm
orcl _ora_19474_mytraceP3.trc
orcl _ora_19474_mytraceP3.trm
orcl _ora_19503_mytraceS3.trc
orcl _ora_19503_mytraceS3.trm
orcl _ora_19534_mytraceP4.trm
orcl _ora_19549_mytraceS4.trc
orcl _ora_19549_mytraceS4.trm
```

```
orcl _ora_19558_mytraceP5.trc
orcl _ora_19558_mytraceP5.trm
orcl _ora_19568_mytraceS5.trc
orcl _ora_19568_mytraceS5.trm
orcl _ora_19583_mytraceP6.trc
orcl _ora_19583_mytraceP6.trm
orcl _ora_19634_mytraceP7.trc
orcl _ora_19634_mytraceP7.trm
$
```

13. 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
$
```

14. 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 19.0.0.0- Development on Wed Jun 3 02:00:00 2018

Copyright (c) 1982, 2019, Oracle and/or its affiliates.  All rights
reserved.
$
```

15. In addition, run `TKPROF` over the trace file that was generated for case 7 (step 12) with the `EXPLAIN` option set to your `TRACE` account.

```
$ tkprof /u01/app/oracle/diag/rdbms/orcl/orcl/trace/*mytraceP7.trc
myreport2.txt explain=trace/trace@orclpdb

TKPROF: Release 19.0.0.0.0- Development on Wed Jan 3 02:00:48 2018

Copyright (c) 1982, 2019, Oracle and/or its affiliates.  All rights
reserved.
$
```

16. After this is done, interpret the trace generated for case 1 (step 6). Case 1 appears near the beginning of the `myreport.txt` file.

    Suggestion: Use a text editor or pager that has a search capability. `vscode`, `vi`, and `less` are available in the class machines. `vscode` provides a GUI.

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

    a. 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.

    b. **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
  ----------------------------------------   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
  ----------------------------------------   Waited  --------------  ------------------
  SQL*Net message to client                     2        0.00          0.00
  SQL*Net message from client                   2        0.00          0.00
********************************************************************
```

17. Interpret the trace generated for case 2 (step 7). 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 clien                      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)
```

```
      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
  ----------------------------------------   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)

********************************************************************
```

18. Interpret the trace generated for case 3 (step 8). This trace starts at the string “update sales set amount_sold=20000”. What do you observe, and what are your conclusions?
    a. 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
  ----------------------------------------   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
  ----------------------------------------   Waited  ----------  ------------
  SQL*Net message to client                       1        0.00          0.00
  SQL*Net message from client                     1        0.01          0.01
********************************************************************************
```

19. Interpret the trace generated for case 4 (step 9). This trace can be found below the string `'sh.sales where amount_sold>0'`. What do you observe, and what are your conclusions?
    a. 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    urrent        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
********************************************************************************
```

20. Interpret the trace generated for case 5 (step 10). This trace can be found using the string `union all`. What do you observe, and what are your conclusions?
    a. 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.
    b. 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      0.04       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
```

```
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,iniexts=: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, bitmapranges=: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: 0949qykm7q1t0
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
  ----------------------------------------   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
********************************************************************************
```

21. Interpret the trace generated for case 6 (step 11). This trace starts at the string `bxraux4u04and`, which is the SQL ID for the statement. What do you observe and what are your conclusions?
    a. 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
  ----------------------------------------   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
********************************************************************************
```

22. Interpret the trace generated for case 7 (step 12). 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
  ----------------------------------------   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
  ----------------------------------------  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
********************************************************************************
```

23. You can now exit your terminal window.

Practices for Lesson 4: Optimizer Fundamentals

Practices for Lesson 4: 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 4-1: 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 ('1999-01','1999-02')
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/orcl/orcl/trace/orcl_ora_7721_MYOPTIM
IZER.trc
Oracle Database 19c Enterprise Edition Release 19.0.0.0.0 -
Production
```

```
Version 19.3.0.0.0 Build label:
RDBMS_19.3.0.0.0DBRU_LINUX.X64_190417
ORACLE_HOME:    /u01/app/oracle/product/19.0.0/dbhome_1
System name:   Linux
Node name:     edvmr1p0
Release: 3.8.13-118.19.7.el6uek.x86_64
Version:  #2 SMP Fri Sep 15 18:08:56 PDT 2017
Machine:  x86_64
VM name:  Xen Version: 4.1 (HVM)
Instance name: orcl
Redo thread mounted by this instance: 1
Oracle process number: 50
Unix process pid: 7721, image: oracle@edvmr1p0 (TNS V1-V3)


*** 2018-01-31T08:15:37.630041+00:00 (ORCLPDB(3))
*** SESSION ID:(29.61112) 2018-01-31T08:15:37.630115+00:00
*** CLIENT ID:() 2018-01-31T08:15:37.630127+00:00
*** SERVICE NAME:(orclpdb) 2018-01-31T08:15:37.630141+00:00
*** MODULE NAME:(sqlplus@edvmr1p0 (TNS V1-V3)) 2018-01-
31T08:15:37.630153+00:00
*** ACTION NAME:() 2018-01-31T08:15:37.630165+00:00
*** CLIENT DRIVER:(SQL*PLUS) 2018-01-31T08:15:37.630175+00:00
*** CONTAINER ID:(3) 2018-01-31T08:15:37.630187+00:00
 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"


*** 2020-06-17 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 ('1999-01','1999-02')
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            = 19.1.0
_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"='1999-01' OR
"T"."CALENDAR_QUARTER_DESC"='1999-02') 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"='1999-01' OR
"T"."CALENDAR_QUARTER_DESC"='1999-02') 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"='1999-01' OR
"T"."CALENDAR_QUARTER_DESC"='1999-02') 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"='1999-01' OR
"T"."CALENDAR_QUARTER_DESC"='1999-02') 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"='1999-01' 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"='1999-01' OR
"T"."CALENDAR_QUARTER_DESC"='1999-02') 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"='1999-01' OR
"T"."CALENDAR_QUARTER_DESC"='1999-02') 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"='1999-01' OR
"T"."CALENDAR_QUARTER_DESC"='1999-02') 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"='1999-01' 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"='1999-01' 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"='1999-01' OR
"T"."CALENDAR_QUARTER_DESC"='1999-02') 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 nbfros=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
    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  resc_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
      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  resc_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  resc_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  resc_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  resc_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  resc_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 frofkksm[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 frofkksm[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"='1999-01' OR
"T"."CALENDAR_QUARTER_DESC"='1999-02'
id=0 frofkksm[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 ('1999-01','1999-02')
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 ('1999-01','1999-02')
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"='1999-01' OR
"T"."CALENDAR_QUARTER_DESC"='1999-02'))
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     :
19.0.00
  parse_schema   : SYS
  plan_hash      : 593420798
  plan_hash_2    : 1128146253
  Outline Data:
  /*+
    BEGIN_OUTLINE_DATA
      IGNORE_OPTIM_EMBEDDED_HINTS
      OPTIMIZER_FEATURES_ENABLE('19.1.0')
      DB_VERSION('19.1.0')
      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            = 19.1.0
…

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
```

Practices for Lesson 5: Generating and Displaying Execution Plans

Practices for Lesson 5: Overview

Practices Overview

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

Practice 5-1: 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. Start SQL Developer, 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 (if the Files tab is not visible, from the menu, select View > Files). Expand ‘My Computer’. ‘Explain_Plan’ folder is shown. If ‘Explain_Plan’ folder is not shown, 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 the `hr` user:

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

```
SELECT last_name, job_id, salary, department_name
FROM employees, departments
WHERE departments.department_id = employees.department_id;
```

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.

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 last_name,  job_id,  salary,  department_name
FROM employees,  departments
WHERE departments.department_id = employees.department_id
and not exists
(select * from jobs where employees.salary between min_salary
and max_salary);
```

a. On the SQL worksheet, enter the preceding query.

b. Click the Explain Plan button.

c. Note the difference in the explain plan.

d. Collapse the information about the filters.

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	9
HASH JOIN			1	9
Access Predicates				
DEPARTMENTS.DEPARTMENT_ID=EMPLOYEES.DEPARTMENT_ID				
NESTED LOOPS			1	9
NESTED LOOPS			1	9
STATISTICS COLLECTOR				
MERGE JOIN		ANTI	1	8
SORT		JOIN	107	4
TABLE ACCESS	EMPLOYEES	FULL	107	3
FILTER				
INDEX	DEPT_ID_PK	UNIQUE SCAN	1	0
Access Predicates				
DEPARTMENTS.DEPARTMENT_ID=EMPLOYEES.DEPARTMENT_ID				
TABLE ACCESS	DEPARTMENTS	BY INDEX ROWID	1	1
TABLE ACCESS	DEPARTMENTS	FULL	1	1

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

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

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

a. In the `hr` connection, click the Autotrace button.

b. Note the Autotrace output tab.

Task

Your environment should be initialized with the et_setup.sql script as shown below before starting next activity:

```
[oracle@b6a3006eeedc Explain_Plan]$ pwd
/home/oracle/labs/solutions/Explain_Plan
[oracle@b6a3006eeedc Explain_Plan]$
[oracle@b6a3006eeedc Explain_Plan]$ sqlplus / as sysdba

SQL*Plus: Release 19.0.0.0.0 - Production on Sun Nov 28 20:38:23 2021
Version 19.3.0.0.0

Copyright (c) 1982, 2019, Oracle.  All rights reserved.


Connected to:
Oracle Database 19c Enterprise Edition Release 19.0.0.0.0 - Production
Version 19.3.0.0.0

SQL> @ep_setup.sql
```

Practice 5-2: 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@edvmr1p0 ~]$ cd /home/oracle/labs/solutions/Explain_Plan/
[oracle@edvmr1p0 Explain_Plan]$ ./ep_startup.sh

SQL*Plus: Release 19.0.0.0.0 - Production on Wed Jun 3 10:01:43 2020
Version 19.3.0.0.0

Copyright (c) 1982, 2019, Oracle.  All rights reserved.

Last Successful login time: Wed Jun 03 2020 08:20:06 +00:00

Connected to:
Oracle Database 19c Enterprise Edition Release 19.0.0.0.0 - Production
Version 19.3.0.0.0

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 19c Enterprise Edition Release 19.0.0.0.0
oduction
Version 19.3.0.0.0
[oracle@edvmr1p0 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> alter session set container = orclpdb;
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@edvmr1p0 Explain_Plan]$ ./ep_session_issue.sh

SQL*Plus: Release 19.0.0.0.0 - Production on Wed Jun 3 10:04:20 2020
Version 19.3.0.0.0

Copyright (c) 1982, 2019, Oracle.  All rights reserved.

Last Successful login time: Wed Jun 03 2020 10:01:43 +00:00

Connected to:
Oracle Database 19c Enterprise Edition Release 19.0.0.0.0 - Production
Version 19.3.0.0.0
```

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 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              :  DONE (ALL ROWS)
 Instance ID         :  1
 Session             :  EP (57:887)
 SQL ID              :  dkz7v96ym42c6
 SQL Execution ID    :  16777219
 Execution Started   :  03/05/2018 08:52:37
 First Refresh Time  :  03/05/2018 08:52:43
 Last Refresh Time   :  03/05/2018 08:52:56
 Duration            :  19s
 Module/Action       :  SQL*Plus/-
 Service             :  orclpdb
 Program             :  sqlplus@[illegible] (TNS V1-V3)
 Fetch Calls         :  1
```

```
Global Stats
=================================================
| Elapsed |   Cpu   |  Other   | Fetch | Buffer |
| Time(s) | Time(s) | Waits(s) | Calls |  Gets  |
=================================================
|      20 |      19 |     0.30 |     1 |   780K |
=================================================

SQL Plan Monitoring Details (Plan Hash Value=1643938535)
=================================================================================================================================
====
| Id |      Operation       |    Name     |  Rows   | Cost |   Time    | Start  | Execs |   Rows   | Activity | Activity Deta
il |
|    |                      |             | (Estim) |      | Active(s) | Active |       | (Actual) |   (%)    |   (# samples)
|
=================================================================================================================================
====
|  0 | SELECT STATEMENT     |             |         |      |        14 |     +6 |     1 |        1 |          |             |
|  1 |   SORT AGGREGATE     |             |         |      |        14 |     +6 |     1 |        1 |     5.26 | Cpu (1)     |
|  2 |    NESTED LOOPS      |             |         |      |        14 |     +6 |     1 |     400M |          |             |
|  3 |     INDEX RANGE SCAN | TEST_C_INDX |         |      |        14 |     +6 |     1 |    20000 |          |             |
|  4 |     INDEX RANGE SCAN | TEST_C_INDX |         |      |        19 |     +1 | 20000 |     400M |    94.74 | Cpu (18)    |
=================================================================================================================================
====


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
```

…

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(*)
----------
 400000000

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@orclpdb
…

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> alter session set container = orclpdb;

Session altered.

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 |  1165  (98)| 00:00:01 |
|   1 |  SORT AGGREGATE        |             |     1 |     6 |            |          |
|*  2 |   HASH JOIN            |             |   400M|  2288M|  1165  (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 (39:18147)
 SQL ID              :  dkz7v96ym42c6
 SQL Execution ID    :  16777221
 Execution Started   :  03/05/2018 09:27:46
 First Refresh Time  :  03/05/2018 09:27:52
 Last Refresh Time   :  03/05/2018 09:27:58
 Duration            :  12s
 Module/Action       :  SQL*Plus/-
 Service             :  orclpdb
 Program             :  sqlplus@edvmr1p0 (TNS V1-V3)

Global Stats
=========================================
| Elapsed |   Cpu   |  Other   | Buffer |
| Time(s) | Time(s) | Waits(s) |  Gets  |
=========================================
|      11 |      11 |     0.18 |     79 |
=========================================
```

```
SQL Plan Monitoring Details (Plan Hash Value=3253233075)
==================================================================================================================================================
| Id   |        Operation         |    Name     |  Rows   | Cost |   Time    | Start  | Execs |   Rows   | Mem | Activity | Activity Detail |
|      |                          |             | (Estim) |      | Active(s) | Active |       | (Actual) |     |   (%)    |   (# samples)   |
==================================================================================================================================================
| -> 0 | SELECT STATEMENT         |             |         |      |         7 |     +6 |     1 |        0 |   . |          |                 |
| -> 1 |   SORT AGGREGATE         |             |       1 |      |         7 |     +6 |     1 |        0 |   . |          |                 |
| -> 2 |    HASH JOIN             |             |    400M | 1165 |        11 |     +2 |     1 |     276M | 2MB |   100.00 | Cpu (11)        |
|    3 |     INDEX FAST FULL SCAN | TEST_C_INDX |   20000 |   12 |         1 |     +6 |     1 |    20000 |   . |          |                 |
| -> 4 |     INDEX FAST FULL SCAN | TEST_C_INDX |   20000 |   12 |         7 |     +6 |     1 |    13824 |   . |          |                 |
==================================================================================================================================================


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 (39:18147)
 SQL ID              :  dkz7v96ym42c6
 SQL Execution ID    :  16777221
 Execution Started   :  03/05/2018 09:27:46
 First Refresh Time  :  03/05/2018 09:27:52
 Last Refresh Time   :  03/05/2018 09:28:00
 Duration            :  16s
 Module/Action       :  SQL*Plus/-
 Service             :  orclpdb
 Program             :  sqlplus@edvmr1p0 (TNS V1-V3)

Global Stats
=========================================
| Elapsed |   Cpu   |  Other   | Buffer |
| Time(s) | Time(s) | Waits(s) |  Gets  |
=========================================
|      13 |      13 |     0.19 |     84 |
=========================================
```

```
SQL Plan Monitoring Details (Plan Hash Value=3253233075)
==========================================================================================================================================
| Id   |        Operation         |     Name     |  Rows   | Cost |   Time    | Start  | Execs |   Rows   | Mem | Activity | Activity Detail |
|      |                          |              | (Estim) |      | Active(s) | Active |       | (Actual) |     |   (%)    |   (# samples)   |
==========================================================================================================================================
| -> 0 | SELECT STATEMENT         |              |         |      |         9 |     +6 |     1 |        0 |   . |          |                 |
| -> 1 |   SORT AGGREGATE         |              |       1 |      |         9 |     +6 |     1 |        0 |   . |          |                 |
| -> 2 |    HASH JOIN             |              |    400M | 1165 |        14 |     +2 |     1 |     328M | 2MB |   100.00 | Cpu (14)        |
|    3 |     INDEX FAST FULL SCAN | TEST_C_INDX  |   20000 |   12 |         1 |     +6 |     1 |    20000 |   . |          |                 |
| -> 4 |     INDEX FAST FULL SCAN | TEST_C_INDX  |   20000 |   12 |         9 |     +6 |     1 |    16384 |   . |          |                 |
==========================================================================================================================================


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 (39:18147)
 SQL ID              :  dkz7v96ym42c6
 SQL Execution ID    :  16777221
 Execution Started   :  03/05/2018 09:27:46
 First Refresh Time  :  03/05/2018 09:27:52
 Last Refresh Time   :  03/05/2018 09:28:03
 Duration            :  17s
 Module/Action       :  SQL*Plus/-
 Service             :  orclpdb
 Program             :  sqlplus@edvmr1p0 (TNS V1-V3)
 Fetch Calls         :  1

Global Stats
=================================================
| Elapsed |   Cpu   |  Other   | Fetch | Buffer |
| Time(s) | Time(s) | Waits(s) | Calls |  Gets  |
=================================================
|      16 |      16 |     0.19 |     1 |     92 |
=================================================

SQL Plan Monitoring Details (Plan Hash Value=3253233075)
==========================================================================================================================================
| Id |        Operation         |     Name     |  Rows   | Cost |   Time    | Start  | Execs |   Rows   |  Mem  | Activity | Activity Detail |
|    |                          |              | (Estim) |      | Active(s) | Active |       | (Actual) | (Max) |   (%)    |   (# samples)   |
==========================================================================================================================================
|  0 | SELECT STATEMENT         |              |         |      |        12 |     +6 |     1 |        1 |     . |          |                 |
|  1 |   SORT AGGREGATE         |              |       1 |      |        12 |     +6 |     1 |        1 |     . |          |                 |
|  2 |    HASH JOIN             |              |    400M | 1165 |        16 |     +2 |     1 |     400M |   2MB |   100.00 | Cpu (16)        |
|  3 |     INDEX FAST FULL SCAN | TEST_C_INDX  |   20000 |   12 |         1 |     +6 |     1 |    20000 |     . |          |                 |
```

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> @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;

  COUNT(*)
----------
 400000000

Elapsed: 00:00:16.27

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

-------------------------------------------------------------------------------------
| Id  | Operation              | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |             |     1 |     6 |  1165  (98)| 00:00:01 |
|   1 |  SORT AGGREGATE        |             |     1 |     6 |            |          |
|*  2 |   HASH JOIN            |             |   400M|  2288M|  1165  (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
          0  physical reads
          0  redo size
        542  bytes sent via SQL*Net to client
        607  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 /*+ gather_plan_statistics */ 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 t1.c=t2.c and t1.c=1;

  COUNT(*)
----------
 400000000

Elapsed: 00:00:16.22
SQL>
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_ID`s 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> @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

dd09kf5dnp1gt       903671040 select sql_id,plan_hash_value,sql_text from v$sql
                              where sql_text like '%from test t1, test t2%'

77xh10nuft3bv      3253233075 EXPLAIN PLAN SET STATEMENT_ID='PLUS6260755' FOR se
                              lect count(*) from test t1, test t2 where t1.c=t2.
                              c and t1.c=1

77xh10nuft3bv      3253233075 EXPLAIN PLAN SET STATEMENT_ID='PLUS6260755' FOR se
                              lect count(*) from test t1, test t2 where t1.c=t2.
                              c and t1.c=1


7 rows selected.

Elapsed: 00:00:00.33
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       |             |       |       |  1165 (100)|          |
|   1 |  SORT AGGREGATE        |             |     1 |     6 |            |          |
|*  2 |   HASH JOIN            |             |   400M|  2288M|  1165  (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)

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: 00:00:00.11
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 |        |       |  1375 (100)|          |      1 |00:00:16.23 |      92 |       |       |          |
|   1 |  SORT AGGREGATE        |             |      1 |      1 |     6 |            |          |      1 |00:00:16.23 |      92 |       |       |          |
|*  2 |   HASH JOIN            |             |      1 |   400M|  2288M|  1375  (99)| 00:00:01 |   400M|00:00:17.30 |      92 |  2757K|  2757K| 1914K (0)|
|*  3 |    INDEX FAST FULL SCAN| TEST_C_INDX |      1 |  20000 | 60000 |    12   (0)| 00:00:01 |  20000 |00:00:00.12 |      46 |       |       |          |
|*  4 |    INDEX FAST FULL SCAN| TEST_C_INDX |      1 |  20000 | 60000 |    12   (0)| 00:00:01 |  20000 |00:00:00.19 |      46 |       |       |          |
--------------------------------------------------------------------------------------------------------------------------------------------------------------

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('19.1.0')
      DB_VERSION('19.1.0')
      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; rowset=1019)
   3 - "T1"."C"[NUMBER,22]
   4 - "T2"."C"[NUMBER,22]
```

16. From session 1, try to retrieve your execution plans from the Automatic Workload Repository. What happens and why?
    a. You can use the previously found `SQL_ID`s 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
------------- --------------------------------------------------
8w580dd6ncgqw select /*+ gather_plan_statistics */ count(*) from
               test t1, test t2 where t1.c=t

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


Elapsed: 00:00:00.04
SQL>
SQL>
```

17. How can you ensure that you retrieve your queries from the Automatic Workload Repository? Implement your solution.
    a. 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> @ep_save_awr
SQL> set echo on
SQL>
SQL> EXEC DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT('ALL');

PL/SQL procedure successfully completed.

Elapsed: 00:00:04.95
SQL>
SQL>
```

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       |             |       |       |  1015 (100)|          |
|   1 |  SORT AGGREGATE        |             |     1 |     6 |            |          |
|   2 |   HASH JOIN            |             |   400M|  2288M|  1015  (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 |
-------------------------------------------------------------------------------------------


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 …
$
```

Practices for Lesson 6: Interpreting Execution Plans and Enhancements

Practices for Lesson 6: Overview

Overview

This practice covers the dynamic plans part of the Adaptive Execution Plans feature in Oracle Database 19c.

Practice 6-1: Using Dynamic Plans

Overview

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

Tasks

1. Start SQL*Plus as `SYS`.

```
$ sqlplus / as sysdba
$ alter session set container = orclpdb;
```

2. Grant the `SELECT ANY DICTIONARY` privilege to `OE`.

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

```
[oracle@edvmr1p0 ~]$ sqlplus / as sysdba

SQL*Plus: Release 19.0.0.0.0 - Production on Wed Jun 3 09:57:54 2020
Version 19.3.0.0.0

Copyright (c) 1982, 2019, Oracle.  All rights reserved.


Connected to:
Oracle Database 19c Enterprise Edition Release 19.0.0.0.0 - Production
Version 19.3.0.0.0

SQL> alter session set container=orclpdb;

Session altered.

SQL> grant select any dictionary to oe;

Grant succeeded.

SQL>
```

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

   SQL> conn oe/oe@orclpdb

```
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;
```

   a. Connect as `oe` with `oe` as the password.

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

Explained.

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

```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------
Plan hash value: 1255158658

-----------------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name                   | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                        |     4 |   128 |     7   (0)| 00:00:01 |
|   1 |  NESTED LOOPS                |                        |     4 |   128 |     7   (0)| 00:00:01 |
|   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")

Note
-----
   - this is an adaptive plan

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

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>
Screws <B.28.S>
Screws <B.28.S>
Screws <B.28.S>
Screws <B.28.S>
Screws <B.28.S>

PRODUCT_NAME
--------------------------------------------------
Screws <B.28.S>
Screws <B.28.S>

13 rows selected.
```

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  axnckp3w6xn6v, child number 0
-------------------------------------
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

Plan hash value: 1553478007

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

PLAN_TABLE_OUTPUT
---------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------
|   0 | SELECT STATEMENT   |                     |       |       |     7 (100)|          |
|*  1 |  HASH JOIN         |                     |     4 |   128 |     7   (0)| 00:00:01 |
|*  2 |   TABLE ACCESS FULL| ORDER_ITEMS         |     4 |    48 |     3   (0)| 00:00:01 |
|   3 |   TABLE ACCESS FULL| PRODUCT_INFORMATION |     1 |    20 |     1   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------

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

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

PLAN_TABLE_OUTPUT
---------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------

Note
-----
   - this is an adaptive plan


27 rows selected.
```

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 oneof 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>
SQL> select sql_id, sql_text, is_resolved_adaptive_plan from v$sql
  2  where sql_text like 'select /*+ monitor*/ product_name%';

SQL_ID        SQL_TEXT                       I
------------- ------------------------------ -
axnckp3w6xn6v 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
```

Practices for Lesson 7: Optimizer: Table and Index Access Paths

Practices for Lesson 7: 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 7-1: 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**
   a. Start SQLDeveloper.
   b. Use the SQL Developer Files tab and open the `$HOME/labs/solutions/Access_Paths/ap_setup.sql` script.

   c. Select `sys_connection`.

d. Execute the script (F5).

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

```
SQL> alter session set container = orclpdb;

Session altered.

SQL> drop table mysales purge;

Table MYSALES dropped.

SQL>
SQL> create table mysales as select * from sh.sales;

Table MYSALES created.

SQL>
SQL> insert into mysales select * from mysales;

918,843 rows inserted.

SQL> commit;

Commit complete.

SQL>
SQL>
SQL> insert into mysales select * from mysales;

1,837,686 rows inserted.
```

```
SQL> commit;

Commit complete.

SQL>
SQL> insert into mysales select * from mysales;

3,675,372 rows inserted.

SQL> commit;

Commit complete.

SQL>
SQL> insert into mysales select * from mysales;

7,350,744 rows inserted.

SQL> commit;

Commit complete.

SQL>
SQL> insert into mysales select * from mysales;

14,701,488 rows inserted.

SQL> commit;
```

```
Commit complete.

SQL>
SQL> insert into mysales values (0,0,sysdate,0,0,0,0);

1 row inserted.

SQL> commit;

Commit complete.

SQL>
SQL> exec dbms_stats.gather_schema_stats('SH');

PL/SQL procedure successfully completed.

SQL>
SQL> create index mysales_prodid_idx on mysales(prod_id) nologging;

Index MYSALES_PRODID_IDX created.

SQL>
SQL> exit;

Disconnected from Oracle Database 19c Enterprise Edition Release 19.0.0.0.0 - Production
Version 19.3.0.0.0
```

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. Drop any indexes available on prod_id column of the MYSALES table.

```
DROP INDEX mysales_prodid_idx;
```

c. Autotrace the query.

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

d. Observe the output.

– 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 3(b) after executing `create_mysales_index.sql`.

   a. Open the `create_mysales_index.sql` file and execute it with `sh_connection`.

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

– 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.

6. **Case 2: Compare Single Column Index Access:** 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`.

```
Run Script in SQL Developer as SH User

drop_customers_indexes.sql
```

Note: Run list_customers_indexes.sql after running above script and delete if you see any indexes manually. But DO NOT delete CUSTOMERS_PK index. Repeat the same steps in next activities as well.

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.

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

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

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`.

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

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
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:** Connect to `sh` and drop all theindexes that are currently created on the `CUSTOMERS` table, except its primary key index, byexecuting `drop_customers_indexes.sql`.

```
Run Script in SQL Developer as SH User

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?

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

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

– 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?

– 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?

- 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:** Connect to `sh` and drop all theindexes that are currently created on the `CUSTOMERS` table, except its primary key index, byexecuting `drop_customers_indexes.sql`.

```
Run Script in SQL Developer as SH User

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.

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

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

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

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:** 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`.

```
Run Script in SQL Developer as SH User

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.

    b. Open and execute the `create_cust_credit_limit_bindex.sql` script.

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.

a. 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';
```

sh_connection~1 | alter_cust_yob_idx.sql | sh_connection~23

SQL Worksheet | History

0.096 seconds

Worksheet | Query Builder

```
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';
```

Script Output

Task completed in 0.096 seconds

```
       NUM NAME                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                
TYPE
---------- ---------- ----------
VALUE
---------- ----------
DISPLAY_VALUE
---------- ----------
DEFAULT_VALUE
---------- ----------
      3784 optimizer_use_invisible_indexes                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                
1
FALSE
FALSE
FALSE



Index CUST_CUST_YEAR_OF_BIRTH_BIDX altered.


INDEX_NAME                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                VISIBILIT
---------- ---------
CUST_CUST_YEAR_OF_BIRTH_BIDX                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                INVISIBLE
CUST_CUST_CREDIT_LIMIT_BIDX                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                VISIBLE
CUSTOMERS_PK                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                VISIBLE
```

26. Autotrace the query in `query07.sql`. What do you observe?

– This is the same query as in step 24. 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`.

```
Run Script in SQL Developer as SH User

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.

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`.

```
Run Script in SQL Developer as SH User

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)`

a. Open and execute the `create_last_name_index.sql` script.

b. Open and execute the `create_first_name_index.sql` script.

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 sql developer. Connect to `sh` and drop allthe indexes that are currently created on the `CUSTOMERS` table, except its primary key index, by executing `drop_customers_indexes.sql`.

```
Run Script in SQL Developer as SH User

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.

35. Autotrace the query in `query11.sql`. What do you observe?

– 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`.

```
Run Script in SQL Developer as SH User

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.

38. Autotrace the query in `query11.sql`. What do you observe?

– The optimizer uses the B*-Tree index; however, this is less efficient compared to the corresponding bitmap index from the previous case in step 35.

39. **Case 10: Function Based Index:** Drop all the `CUSTOMERS` indexes, except its primary key index. 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`.

```
Run Script in SQL Developer as SH User

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)`

41. Autotrace the query in `query12.sql`. What do you observe?

– 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.

    a. You can create a function-based index by using `create_lower_cust_last_name_index.sql`.

43. Check if your solution executes faster than in the case of the query in step 41.

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.

45. Autotrace the query in `query13a.sql` and `query13b.sql`. What do you observe?

– 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?

– The optimizer directly uses the index-organized structure, which is extremely efficient in this case compared to the previous step.

47. Open and execute the `iot_cleanup.sql` script to clean up your environment.

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.

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.

Practices for Lesson 8:
Optimizer: Join Operators

Practices for Lesson 8: Overview

Practices Overview

In these practices, you will examine three SQL statements and compare the different join methods for each statement.

Practice 8: 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. Start SQLDevloper.
2. Open and execute the `join_setup.sql` script by using `sys_connection`.

3. Open the connection named `hr` 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: `hr_connection`
   Username: `hr`
   Password: `hr`
   Select Save Password.
   Service Name: `orclpdb`
   Click Test.
   Click Connect.

4. **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.

   a. Enter the following query or open `small_tables_join.sql` as the `hr` user:

```
select department_name, d.location_id, l.location_id,
l.street_address
from departments d, locations l
where d.location_id = l.location_id and department_name like
'S%';
```

   Run Autotrace twice to load the buffer cache. Observe the statistics in the second run. What do you observe?

   b. 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.

   c. 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(D L) */ department_name, d.location_id,
l.location_id, l.street_address
from departments d, locations l
where d.location_id = l.location_id
and department_name like 'S%';
```

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(D L) */ department_name, d.location_id,
l.location_id, l.street_address
from departments d, locations l
where d.location_id = l.location_id
and department_name like 'S%';
```

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.

5. 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.
      Service Name: `orclpdb`
      Click Test.
      Click Connect.

6. **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.

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.

7. 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.
Service Name: `orclpdb`
Click Test.
Click Connect.

8. **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 p.prod_name, p.prod_id
FROM sh.products p, sh.sales s
where p.prod_id = s.prod_id
and s.prod_id < 100;
```

   Run Autotrace twice to warm the buffer cache.

   What do you observe?

   Notice the consistent gets and the estimated cost.

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(p s) */ p.prod_name, p.prod_id
FROM sh.products p, sh.sales s
where p.prod_id = s.prod_id
and s.prod_id < 100 ;
```

Run Autotrace. What do you observe?

The consistent gets and estimated cost values are much higher than the values for the hash join method.

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(p s) */ p.prod_name, p.prod_id
FROM sh.products p, sh.sales s
where p.prod_id = s.prod_id
and s.prod_id < 100 ;
```

Run Autotrace. What do you observe?

The estimated cost is higher than the hash join and nested loops method, but the consistent gets is lower than the nested loops method.

Practices for Lesson 9: Other Optimizer Operators

Practices for Lesson 9: Overview

Practices Overview

In these practices, you use the Results Cache to improve SQL performance with repeated queries and examine the SQL statements that use other access paths.

Practice 9-1: 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:

```
[oracle@edvmr1p0 Query_Result_Cache]$ ./result_cache_setup.sh
…

#!/bin/bash

cd /home/oracle/labs/solutions/Query_Result_Cache

sqlplus sys/fenago as sysdba <<FIN!

set echo on
alter session set container = orclpdb;

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@orclpdb

exec dbms_result_cache.flush;

drop table cachejfv purge;

create table cachejfv(c varchar2(500)) tablespace users;

insert into cachejfv
values('aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa');
insert into cachejfv select * from cachejfv;
insert into cachejfv select * from cachejfv;
insert into cachejfv select * from cachejfv;
insert into cachejfv select * from cachejfv;
```

```
insert into cachejfv select * from cachejfv;
insert into cachejfv select * from cachejfv;
insert into cachejfv select * from cachejfv;
insert into cachejfv select * from cachejfv;
insert into cachejfv select * from cachejfv;
insert into cachejfv select * from cachejfv;
insert into cachejfv select * from cachejfv;
insert into cachejfv select * from cachejfv;
insert into cachejfv select * from cachejfv;
insert into cachejfv select * from cachejfv;
insert into cachejfv select * from cachejfv;
insert into cachejfv select * from cachejfv;
insert into cachejfv select * from cachejfv;
insert into cachejfv select * from cachejfv;
insert into cachejfv select * from cachejfv;
insert into cachejfv select * from cachejfv;
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. Start SQLDeveloper.
3. 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.”
      Service Name: `orclpdb`

Click Test.
Click Connect.

4. 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 by using the `sys` user and then execute the `check_result_cache.sql` script again by using the `qrc` user.

```
begin
dbms_result_cache.flush;
end;
```

Disconnect from the `SYS` connection.

5. 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';
```

6. Determine the execution plan of the query in `query1.sql`. What do you observe?

Script Output | Explain Plan

SQL | 0.224 seconds

OPERATION	OBJECT_NAME	OPTIONS	CAR...	COST
SELECT STATEMENT			1	18446744...
RESULT CACHE	4js8kjft2ytpk4mr667spmtv78			
SORT		AGGREGATE	1	
MERGE JOIN		CARTESIAN	1844...	18446744...
MERGE JOIN		CARTESIAN	1844...	18446744...
MERGE JOIN		CARTESIAN	1844...	18446744...
MERGE JOIN		CARTESIAN	1844...	18446744...
MERGE JOIN		CARTESIAN	1152...	84935652...
MERGE JOIN		CARTESIAN	1099...	8100095063
TABLE ACCESS	CACHEJFV	FULL	1048...	7727
Filter Predicates				
C7.C='b'				
BUFFER		SORT	1048...	8100087336
TABLE ACC	CACHEJFV	FULL	1048...	7725
Filter Predicates				
C6.C='b'				
BUFFER		SORT	1048...	84935652...
TABLE ACCESS	CACHEJFV	FULL	1048...	7725
Filter Predicates				
C5.C='b'				
BUFFER		SORT	1048...	18446744...
TABLE ACCESS	CACHEJFV	FULL	1048...	7725
Filter Predicates				
C4.C='b'				
BUFFER		SORT	1048...	18446744...
TABLE ACCESS	CACHEJFV	FULL	1048...	7725
Filter Predicates				
C3.C='b'				
BUFFER		SORT	1048...	18446744...
TABLE ACCESS	CACHEJFV	FULL	1048...	7725
Filter Predicates				
C2.C='b'				
BUFFER		SORT	1048...	18446744...
TABLE ACCESS	CACHEJFV	FULL	1048...	7725
Filter Predicates				
C1.C='b'				

– Because of the `result_cache` hint, the result of the query is computed using the result cache.

7. 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;
```

– You can now see that the result of your query is cached.

8. Flush the buffer cache of your instance.

```
alter system flush buffer_cache;
```

9. Rerun `query1.sql`. What do you observe?

– The execution time for the query is now almost instantaneous.

10. Insert a new row into the `CACHEJFV` table by using the following statement:

```
insert into cachejfv values('c');
commit;
```

11. 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;
```

– The corresponding result cache entry is automatically invalidated.

12. Rerun `query1.sql`.

13. 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;
```

– Again, it takes some time to execute the query. The result cache shows that a new entry has been added for the new result.

14. Generate a detailed result cache memory report by entering the following commands:

```
set serveroutput on
EXEC DBMS_RESULT_CACHE.MEMORY_REPORT(detailed=>true);
```

15. Rerun the `query1.sql` script. What do you observe?

– The query again uses the result that was previously cached.

16. Ensure that you bypass the result cache before performing the next step.

```
exec DBMS_RESULT_CACHE.BYPASS(bypass_mode=>true);
```

17. Rerun `query1.sql`. What do you observe?

– The query again takes longer to execute because it no longer uses the result cache.

18. Ensure that you no longer bypass the result cache and check whether your query uses it again.

    a. Execute the following statement:

```
exec DBMS_RESULT_CACHE.BYPASS(bypass_mode=>false);
```

    b. Rerun `query1.sql`.

19. 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?

- 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.

20. How would you force the previous query to use the cached result without using hints?
    a. 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;
```

Start Page | qrc | check_result_cache.sql | query1.sql | query2.sql | force_result_cache.sql

SQL Worksheet | History

0.097 seconds

qrc

Worksheet | Query Builder

```
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;
```

Script Output

Task completed in 0.097 seconds

```
SQL> set echo on
SQL>
SQL> show parameter result_cache_mode
NAME                 TYPE   VALUE
---------------- ------ ------
result_cache_mode string MANUAL
SQL>
SQL> select type,status,name,object_no,row_count,row_size_avg from v$result_cache_objects order by 1;

TYPE       STATUS    NAME
---------- --------- ------------------------------------------------------------------------------------------
Dependency Published SYS.WRI$_ADV_TASKS
Dependency Published QRC.CACHEJFV
Dependency Published SYS.WRI$_ADV_EXECUTIONS
Dependency Published SYS.DBA_ADVISOR_TASKS
Result     Published SELECT /*+ NO_STATEMENT_QUEUING RESULT_CACHE (SYSOBJ=TRUE) */ ADVISOR_NAME,EXECUTION_END FROM "SY
Result     Published SELECT  /*+ result_cache q_name(Q1) */
                       COUNT(*)
                     FROM cachejfv c1, cachejfv c2, cachejfv c3, cachejfv c4, cachejfv c5,
                       cachej

Result     Invalid   SELECT  /*+ result_cache q_name(Q1) */

TYPE       STATUS    NAME
---------- --------- ------------------------------------------------------------------------------------------
                       COUNT(*)
                     FROM cachejfv c1, cachejfv c2, cachejfv c3, cachejfv c4, cachejfv c5,
                       cachej


7 rows selected.

SQL>
SQL> alter session set result_cache_mode=force;

Session altered.
```

b. View the explain plan for the `query2.sql` script.

c. Rerun the `query2.sql` script to verify that the query runs instantaneously because you successfully used the cached result.

d. Finally, undo your change.

```
alter session set result_cache_mode=manual;
```

21. Clear the result cache. Query `V$RESULT_CACHE_OBJECTS` to verify the clear operation.

```
exec dbms_result_cache.flush;
```

22. Open and execute the `cre_func.sql` script. This script creates a PL/SQL function that uses the result cache.

23. 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;
```

24. Call the new function with `'b'` as its argument. What do you observe?

```
select cachejfv_count('b') from dual;
```

– 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.

25. Call the new function with `'b'` as its argument again. What do you observe?

```
select cachejfv_count('b') from dual;
```

– This time the function executes almost instantaneously.

26. Call the new function again, but with `'c'` as its argument. What do you observe?

– 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 9-2: 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.”
      Service Name: `orclpdb`
      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 sys/fenago as sysdba
alter session set container =orclpdb;

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@orclpdb

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('SHC');

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.”
      Service Name: `orclpdb`
      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.

6. Use the `shc` connection to autotrace the query in the `query15b.sql` script. What do you observe?

– The optimizer decides to use the cluster access path to retrieve data. The cost is minimal.

7. Execute the `query15a.sql` script again.

8. Open and autotrace the query in the `query16.sql` script as the `SHC` user. What do you observe?

– 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.

10. Autotrace the query in the `query17.sql` script. What do you observe?

– 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.

12. Open and autotrace the query in the `query18.sql` script as the `SHC` user. What do you observe?

– 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.

14. Autotrace the query in the `query19.sql` script. What do you observe?

– 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 sys/fenago as sysdba
alter session set container = orclpdb;

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@orclpdb

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('NIC');


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.”

       Service Name: `orclpdb`

       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.

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.

    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 sys/fenago as sysdba
alter session set container = orclpdb;


DROP user IC cascade;

CREATE USER ic IDENTIFIED BY ic;

GRANT DBA TO ic;
GRANT SELECT_CATALOG_ROLE TO ic;
GRANT SELECT ANY DICTIONARY TO ic;

connect ic/ic@orclpdb

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('IC');

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:
icPassword: ic
Select "Save Password."
Service Name: orclpdbClick
Test.
Click Connect.

21. Open and execute the `nic_query_a.sql` script as the `ic` user.

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.

23. **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.

24. 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: `Ignore the OLAP errors.`

```
$ cd /home/oracle/labs/solutions/Access_Paths/
$ ./ap_cleanup.sh
$ ORACLE_SID = [orcl]?

…
Connected to:…
```

```
SQL>
SQL> @ORACLE_HOME/demo/schema/sales_history/sh_main sh users
temp fenago
/u01/app/oracle/product/12.2.0/dbhome_1/demo/schema/sales_histor
y/ /home/oracle/ v3 localhost:1521/orclpdb
SQL> Rem
SQL> Rem $Header: rdbms/demo/schema/sales_history/sh_main.sql
/main/12 2015/03/19 10:23:26 smtaylor Exp $
SQL> Rem
SQL> Rem sh_main.sql
SQL> Rem
SQL> Rem Copyright (c) 2001, 2015, Oracle and/or its affiliates.
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     smtaylor          03/19/15 - added parameter 8,
connect_string
SQL> Rem     smtaylor           03/19/15 - added
@&connect_string to CONNECT
```

```
SQL> Rem     smtaylor           03/19/15 - added pararmeter
&connect_string to script calls
SQL> Rem        awesley         04/03/12 - Remove cwm_user
SQL> Rem        jmadduku        02/18/11 - Grant Unlimited
Tablespace priv with RESOURCE
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 - depricating 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:

specify connect string as parameter 8:

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 10: Introduction to Optimizer Statistics Concepts

Practices for Lesson 10: 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 10-1: 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. Start SQLDeveloper.
2. Change directories to `$HOME/labs/solutions/Optimizer_Statistics`. Start a SQL Developer session, connect to the database as the `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';
```

Note: If SH connection isn't there, then create a one by using the following specifications: Name: sh
Username: sh
Password: sh
Select "Save Password." Service Name: orclpdb Click Test.
Click Connect.

3. Drop all the `CUSTOMERS` indexes, except its primary key index. 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`script. And create an index on the `customers.cust_last_name` column. Use the `opt_2.sql` file.

```
Run Script in SQL Developer as SH User

drop_customers_indexes.sql
```

4. 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

```
CREATE INDEX CUSTOMERS_LAST_NAME_IDX ON
customers(cust_last_name);
```

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';
```

5. 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;
```

...

6. Gather statistics on the `CUSTOMERS3` table. Use the `opt_5.sql` file.

```
EXEC DBMS_STATS.GATHER_TABLE_STATS(null,'CUSTOMERS3');
```

7. 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';
```

8. 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);
```

9. 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';
```

Note: The `CUSTOMERS3` table 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.

10. 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';
```

11. Execute the following query to display the cursor for the query. Use the `opt_10.sql` file.

```
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY_CURSOR());
```

12. 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';
```

13. Execute the following query to display the cursor for the query. Use the `opt_12.sql` file.

```
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY_CURSOR());
```

Practice 10-2: 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);
```

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;
/
```

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

```
COL EXTENSION_NAME FORMAT a30
COL EXTENSION FORMAT a35

SELECT EXTENSION_NAME, EXTENSION
FROM   USER_STAT_EXTENSIONS
WHERE  TABLE_NAME='CUSTOMERS';
```

opt_17.sql file:

6. 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';
```

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_stat
e_province))');
END;
/
```

8. Drop all the `CUSTOMERS` indexes, except its primary key index. Connect to `sh` and drop all the indexes that are currently created on the `CUSTOMERS` table, except its primary key index, by executing the `drop_customers_indexes.sql` script. And create an index on the `customers.cust_last_name` column. Use the `opt_2.sql` file.

```
Run Script in SQL Developer as SH User

drop_customers_indexes.sql
```

Practice 10-3: 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. Note: 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

alter session set container = orclpdb;
-----------------------------------------------------------------------------------

set echo on

drop user ags cascade;

create user ags identified by ags;

grant dba to ags;

connect ags/ags@orclpdb

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>alter session set container = orclpdb;
Session 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@orclpdb

…

SQL>
```

4. Create a new connection in SQL Developer for the `ags` user with the following properties:

   Name: `ags`

   `User: ags`

   `Password: ags`

   `Service Name: orclpdb`

   `Click`

   `Test.`

   `Click`

   `Connect.`

5. Using `ags`, determine the distribution of the `deptno` values found in the `EMP` table. What do you observe?

   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.

6. Determine if there are histograms available on any column of the `EMP` table. What do you observe?

   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.

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.

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.

	INDEX_OWNER	INDEX_NAME	UNIQUENESS	STATUS	INDEX_TYPE	TEMPORARY	PARTITIONED	FUNCIDX_STATUS	JOIN_INDEX	COLU
1	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.

	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)
25	SCOPE	SHARED

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 using `ags`. You see that for `deptno = 9`, the optimizer decided to use the index.

b. Open the `literal10.sql` file and autotrace using `ags`. 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 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.

11. Turn off dynamic statistics.

    a. Set the parameter for your session to 0. Use the following command.

    ```
    ALTER session SET optimizer_dynamic_sampling = 0;
    ```

b. Confirm that the parameter has changed. Open `SQL_HISTORY` , select the `select * from V$parameter command, click` **Replace**, `and execute.`

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;
```

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.

15. Verify that you modified the statistics of the `EMP` table correctly.

    In the navigator pane, select the EMP table under ags. On the EMP tab, click the Statistics subtab.

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.

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.

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.

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.
    a. In the navigator pane, select the `EMP` table and click the Statistics subtab on the `EMP` tab, and refresh.

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	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)
25	SCOPE	SHARED

22. How would you determine the list of automated tasks that exist on your database?

a. 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. Run the following command from the terminal as the `sys` user:

```
alter system set "_enable_automatic_maintenance"=1;
```

```
[oracle@edvmr1p0 Automatic_Gather_Stats]$ sqlplus / as sysdba

SQL*Plus: Release 19.0.0.0.0 - Production on Wed Jun 3 12:27:24 2020
Version 19.3.0.0.0

Copyright (c) 1982, 2019, Oracle.  All rights reserved.


Connected to:
Oracle Database 19c Enterprise Edition Release 19.0.0.0.0 - Production
Version 19.3.0.0.0

SQL> alter system set "_enable_automatic_maintenance"=1;

System altered.
```

24. From SQL Developer, open and execute the `run_ags_pl.sql` script. This script forces the execution of the automatic statistics gathering task.
    a. If you do not already have a sys_connection, create a connection as the `sysdba` user.

       Name: `sys_connection`

       `Username: sys`

       `Password: fenago`

       `Role: sysdba`

       `Service Name: orclpdb`
    b. Test and save the connection.
    c. Open the `run_ags_pl.sql` file.

d. Open the Dbms Output pane (View > Dbms Output) and click the green plus sign to connect to `sys_connection`.

e. Execute the `run_ags_pl.sql` script as the `sys` user. Use the `sys_connection` connection. This script takes two minutes to run.

f. Close the Dbms Output pane.

25. 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. 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
1	NUM_ROWS	100000
2	BLOCKS	874
3	AVG_ROW_LEN	50
4	SAMPLE_SIZE	100000
5	LAST_ANALYZED	08-MAR-18
6	LAST_ANALYZED_SINCE	08-MAR-18

	OWNER	TABLE_NAME	COLUMN_NAME	NUM_DISTINCT	LOW_VALUE	HIGH_VALUE
1	AGS	EMP	EMPNO	100000	C102	C30B
2	AGS	EMP	ENAME	100000	4141414449465054595 54B4C464F4E4A45465348	5A5A5A524E5448484346434 24A4
3	AGS	EMP	PHONE	100000	414141474D56584C484D4454425A59414D585452	5A5A5A57454A435452474757595
4	AGS	EMP	DEPTNO	11	80	C10B

26. 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.

`Note: The output is different in literal10.sql AUTOTRACE, it doesn't use index. Please, verify once.`

27. Close the `ags` connection and the file tabs in SQL Developer.
28. 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 10-4: 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/labs/solutions/System_Stats directory.

Tasks

1. The sysstats_setup.sh script located in your $HOME/labs/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

```
#!/bin/bash

cd /home/oracle/labs/solutions/System_Stats

sqlplus / as sysdba @sysstats_setup.sql
-------------------------------------------------------------------------------------



set echo on

connect sys/fenago as  sysdba; alter
session set container = orclpdb;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@orclpdb

drop table t purge;

drop table z purge;

create table t(c number);

insert into t values (1);
```

as follows:

```
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/labs/solutions/System_Stats
$ ./sysstats_start.sh

…
SQL> alter session set container = orclpdb;
Session altered.

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@orclpdb


…

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?

   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> alter session set container = orclpdb;
Session altered.
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?

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,

```
SQL> connect jfv/jfv@orclpdb
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
|
```

and the execution plan now includes CPU costing information.

```
-----------------------------------------------------------------------------------------------
------
|   0 | SELECT STATEMENT    |         |     |     |   123 (100)|    |
|   1 |  SORT AGGREGATE     |         |   1 |   6 |        |        |

PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------------------------------
--------------
|*  2 |   HASH JOIN        |    |   524K|  3072K|   123   (3)|
00:00:01 |
|   3 |    INDEX FULL SCAN  | IZ   |   100 |   300 |  1   (0)|
00:00:01 |
|   4 |    TABLE ACCESS FULL| T    |   524K|  1536K|   121   (2)|
00:00:01 |
-----------------------------------------------------------------------------------------------
---------

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$sql_plan where address in (select address
  4                                                  from v$sql
  5                                                  where sql_text like
'%system stats%' and
  6                                                      sql_text not like
'%connect%'));

OPERATIONS      OBJECT_NAME OPTIONS       COST_CPU_IO
--------------------------- --------------- -------------------- ------------------------------
----------------------------------------
SELECT STATEMENT                     123 --  --
SORT                    AGGREGATE     --  --
HASH JOIN                            123 -- 137938260 -- 120
INDEX                 IZ      FULL SCAN   1 -- 27121 -- 1
TABLE ACCESS          T             FULL        121 -- 84867339 -- 119
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> alter session set container = orclpdb;

Session altered.
SQL>
SQL> drop user jfv cascade;

User dropped.

SQL>
SQL> alter system flush shared_pool;

System altered.

SQL>
SQL> exit;
```

Practices for Lesson 11:
Using Bind Variables

Practices for Lesson 11: 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 11-1: Using Adaptive Cursor Sharing

Overview

In this practice, you experiment with bind variable peeking and adaptive cursor sharing.

Tasks

1. **Note:** 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:

```
#!/bin/bash

cd /home/oracle/labs/solutions/Adaptive_Cursor_Sharing

sqlplus / as sysdba @acs_setup.sql
--------------------------------------------------------------------------------
set echo on
alter session set container = orclpdb;
drop user acs cascade;

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

grant dba, connect to acs;

connect acs/acs@orclpdb

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;
```

2. 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@orclpdb
…
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>
```

3. 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?

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             |         |     1 |    16 |
|          |
|   2 |   TABLE ACCESS BY INDEX ROWID| EMP     |    10 |   160 |     2
(0)| 00:00:01 |
```

```
|*  3 |    INDEX RANGE SCAN       | EMP_I1 |    10 |       |     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?

The optimizer uses a full table scan because value 10 is not a

```
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
```

selective value.

```
--------------------------------------------------
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   |   |     |     |   120 (100)|    |
|   1 |  SORT AGGREGATE    |   |   1 |    16 |         |      |
|*  2 |   TABLE ACCESS FULL| EMP  | 99900 |  1560K|   120   (1)|
00:00:02 |
-----------------------------------------------------------------------------------------------
--------

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

```
SQL> variable deptno number;
SQL> exec :deptno := 9

PL/SQL procedure successfully completed.

SQL> @select_deptno_bind
SQL> set echo on
SQL>
```

used.

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

```
   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?

   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

```
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>
```

   path that was used.

9. Perform steps 7 and 8 again, but this time using 10 as the bind value for DEPTNO. What do you observe and why?

   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    |    10 |   160 |     2
     (0)| 00:00:01 |
|*  3 |    INDEX RANGE SCAN       | EMP_I1 |    10 |       |     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?

    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   |   |     |     |   120 (100)|    |
|   1 |  SORT AGGREGATE    |   |   1 |    16 |         |      |
|*  2 |   TABLE ACCESS FULL| EMP  | 99900 |  1560K|   120   (1)|
00:00:02 |
-----------------------------------------------------------------------------------------------
--------

Peeked Binds (identified by position):
-------------------------------------------------

   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         878

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

```
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
```

same child cursor.

```
Plan hash value: 2083865914

-----------------------------------------------------------------------------------------------
---
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time
|
-----------------------------------------------------------------------------------------------
---------
|   0 | SELECT STATEMENT   |   |     |     |   120 (100)|    |
|   1 |  SORT AGGREGATE    |   |   1 |    16 |         |      |
|*  2 |   TABLE ACCESS FULL| EMP  | 99900 |  1560K|   120   (1)|
00:00:02 |
-----------------------------------------------------------------------------------------------
--------

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         1684
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="12.2.0.1";

Session altered.

SQL> alter system flush shared_pool;

System altered.

SQL>
```

14. Perform step 12 again. What do you observe and why?

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: 588610755

-----------------------------------------------------------------------------------------------
-----------------------------------------------
| Id  | Operation                          | Name   | Rows  | Bytes |
Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
-----------------------------------------------
|   0 | SELECT STATEMENT                   |        |       |       |
2 (100)|          |
|   1 |  SORT AGGREGATE                    |        |     1 |    16 |
       |          |
|   2 |   TABLE ACCESS BY INDEX ROWID BATCHED| EMP    |    10 |   160
|     2    (0)| 00:00:01 |
|*  3 |    INDEX RANGE SCAN                | EMP_I1 |    10 |       |
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         72

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: 588610755
-------------------------------------------------------------------------------------------

-----------------------------------------------
| Id  | Operation                      | Name   | Rows  | Bytes |
Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
-----------------------------------------------
|   0 | SELECT STATEMENT               |        |       |       |
2 (100)|          |
|   1 |  SORT AGGREGATE                |        |     1 |    16 |
         |          |
```

```
|   2 |   TABLE ACCESS BY INDEX ROWID BATCHED| EMP    |    10 |   160
|     2    (0)| 00:00:01 |
|*  3 |    INDEX RANGE SCAN              | EMP_I1 |    10 |       |
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        1026
```

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

```
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;
```

is used to better handle your statement.

```
  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   |   |     |     |   120 (100)|    |
|   1 |  SORT AGGREGATE    |   |   1 |    16 |         |      |
|*  2 |   TABLE ACCESS FULL| EMP  | 99900 |  1560K|   120   (1)|
00:00:01 |
--------------------------------------------------------------------------------------------------
--------

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        1026
           1          1         809

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 11-2: Using CURSOR_SHARING (Optional)

In this practice, you investigate the use of the CURSOR_SHARING initialization parameter.

1. **Note:** 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

```
#!/bin/bash

cd /home/oracle/labs/solutions/Cursor_Sharing

sqlplus / as sysdba @cs_setup.sql

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

set echo on
alter session set container = orclpdb;
drop user cs cascade;

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

grant dba, connect to cs;

connect cs/cs@orclpdb

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),
```

listed as follows:

```
         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 the 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

```
set linesize 200 pagesize 1000
```

command:

```
$ cd $HOME/labs/solutions/Cursor_Sharing
$ sqlplus cs/cs@orclpdb
…
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?

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>
```

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> @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
---------------- ---------- ---------- ---------------- --------------- ------
000000006EF41E20 3434097775          0 SELECT STATEMENT
000000006EF41E20 3434097775          0 SORT             AGGREGATE
000000006EF41E20 3434097775          0 TABLE ACCESS     BY INDEX ROWID  EMP
                                                        BATCHED

000000006EF41E20 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?

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

```
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
```

FORCE option for the CURSOR_SHARING initialization parameter.

```
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 …

$
```

Practices for Lesson 12: SQL Plan Management

Practices for Lesson 12: 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 12-1: 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 preceding practices.

```
$ cd /home/oracle/labs/solutions/SPM
$ sqlplus / as sysdba
SQL> alter session set container = orclpdb;
Session altered.
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@orclpdb

-- 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> alter session set optimizer_mode = all_rows;
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 and execute twice.

```
SQL> @query1.sql

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

no rows selected

SQL> @query1.sql

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

no rows selected
```

4. An initial plan baseline is automatically created for a new statement. 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>
```

5. 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

```
SQL> alter session set optimizer_mode = first_rows;

Session altered.

-- Execute the query1.sql

SQL> @query1.sql
SQL> set echo on
SQL>
```

execution.

```
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 YES  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 NO  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.

6. 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.
```

7. 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

8. 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
```

9. 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>
```

10. Reset the optimizer mode to `ALL_ROWS` optimization.

```
-- Execute the below Statement

SQL> alter session set optimizer_mode = all_rows;

Session altered.
```

11. 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>
```

12. 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>
```

13. 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>
```

14. 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>
```

15. 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
```

16. 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

17. 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>
```

18. Now, 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.

-- 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>
```

19. Reset the optimizer mode to `ALL_ROWS`.

```
-- Execute the below Statement

SQL> alter session set optimizer_mode = all_rows;

Session altered.
```

20. 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>
```

21. 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
select /*LOAD_CC*/ * from sh.sales
where quantity_sold > 40 order by prod_id
SQL_PLAN_gdkvzfhrgkda711df68d0 MANUAL-LOAD-    YES YES NO  YES
                                FROM-CURSOR-CACHE
1.7783E+19 SQL_f6cb7f742ef93547
select /*LOAD_CC*/ * from sh.sales
where quantity_sold > 40 order by prod_id
SQL_PLAN_gdkvzfhrgkda754bc8843 MANUAL-LOAD    YES YES NO  YES
                                FROM-CURSOR-CACHE

SQL>
-- You should see two accepted baselines
```

22. 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

23. 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>
```

24. 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>
```

25. Reset the optimizer mode to `ALL_ROWS`.

```
-- Execute the below Statement

SQL> alter session set optimizer_mode = all_rows;

Session altered.
```

26. 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_0p7c74s0ftt2p54bc8843 MANUAL-LOAD    YES YES NO  YES
                               FROM-CURSOR-CACHE

7.6492E+17 SQL_0a9d872600ece455
select /*SPM_USE*/ * from sh.sales
where quantity_sold > 40 order by prod_id
SQL_PLAN_0p7c74s0ftt2pce885bc0 MANUAL-LOAD    YES YES NO  YES
                               FROM-CURSOR-CACHE



SQL>
SQL>
```

27. 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.

```
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.
```

Use the explain_query4_note.sql script.

```
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>
```

28. 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

```
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*/%';
```

    baseline.

```
SQL_HANDLE                      PLAN_NAME
---------------------------------------- ----------------------------------------
SQL_0a9d872600ece455       SQL_PLAN_0p7c74s0ftt2p54bc8843
SQL_0a9d872600ece455       SQL_PLAN_0p7c74s0ftt2pce885bc0

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_0p7c74s0ftt2p54bc8843 MANUAL-LOAD-
                               FROM-CURSOR-CACHE 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_0p7c74s0ftt2pce885bc0 MANUAL-LOAD-
                               FROM-CURSOR-CACHE YES 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_0p7c74s0ftt2pce885bc0" used
for this statement

16 rows selected.
```

29. 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

```
SQL> @check_baseline_used2.sql
SQL> set echo on
SQL>
SQL> variable cnt number;
SQL>
SQL> exec :cnt := dbms_spm.alter_sql_plan_baseline( -
```

the plan stating that a baseline was used.

```
>       sql_handle        => 'SQL_0a9d872600ece455', -
>       plan_name         => 'SQL_PLAN_0p7c74s0ftt2pce885bc0', -
>       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_0p7c74s0ftt2p54bc8843  MANUAL-LOAD-
                                FROM-CURSOR-CACHE 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_0p7c74s0ftt2pce885bc0  MANUAL-LOAD-
                                FROM-CURSOR-CACHE 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>
```

30. 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>
```

31. 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 loop join.

```
SQL> @check_evolve_plan.sql
SQL> set echo on
SQL>
SQL> explain plan for
  2  select /*+ USE_NL(s c) FULL(s) FULL(c) */
  3          c.cust_id, sum(s.quantity_sold)
  4  from sh.sales s, sh.customers c
  5  where s.cust_id = c.cust_id and c.cust_id < 2
  6  group by c.cust_id;

Explained.

SQL>
SQL> select * from table(dbms_xplan.display(null, null));

PLAN_TABLE_OUTPUT--------------------------------------------------------------------
----------------------

Plan hash value: 212151715
```

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

| Id  | Operation          | Name          | Rows | Bytes | Cost
(%CPU)| Time
    | Pstart| Pstop |

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

|   0 | SELECT STATEMENT   |          |   1 |  23 | 471   (2)| 00:00
:01 |         |     |

|   1 |  HASH GROUP BY          |          |   1 |  23 | 471   (2)|
00:00
:01 |         |     |

|   2 |   NESTED LOOPS          |          |   1 |  23 | 471   (2)|
00:00
:01 |         |     |

|   3 |    VIEW            | VW_GBC_5  |      1 |  18 | 260   (3)|
00:00
:01 |         |     |

|   4 |     HASH GROUP BY |          |   1 |   8 | 260   (3)| 00:00
:01 |         |     |

|   5 |      PARTITION RANGE ALL|         | 130 |  1040 |    259
(2)| 00:00

PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------------------------
--------------------
:01 |       1 |  28 |

|*  6 |       TABLE ACCESS FULL | SALES     |   130 |  1040 |
    259   (2)| 00:00
:01 |       1 |  28 |

|*  7 |    TABLE ACCESS FULL    | CUSTOMERS |      1 |  5 | 211
(1)| 00:00
```

```
:01 |         |     |

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


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


   6 - filter("S"."CUST_ID"<2)
   7 - filter("C"."CUST_ID"<2 AND "ITEM_1"="C"."CUST_ID")

20 rows selected.
SQL>
```

32. 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

```
SQL> @load_evolve_baseline
SQL> set echo on
SQL>
SQL> variable cnt number;
SQL>
SQL> variable sqlid varchar2(20);
SQL>
SQL> select /*+ USE_NL(s c) FULL(s) FULL(c) */
  2         c.cust_id, sum(s.quantity_sold)
  3  from sh.sales s, sh.customers c
  4  where s.cust_id = c.cust_id and c.cust_id < 2
  5  group by c.cust_id;

no rows selected

SQL>
SQL> begin
  2   select sql_id into :sqlid from v$sql
  3   where sql_text like 'select /*+ USE_NL(s c) FULL(s)
FULL(c) */%';
  4  end;
  5  /
```

fixed.

```
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_18a1ef14c17f5b75
select /*+ USE_NL(s c) FULL(s) FULL(c) */
       c.cust_id, sum(s.quantity_sold)
SQL_PLAN_1j8gg2m0ryqvpa496391d  MANUAL-LOAD-
                                FROM-CURSOR-CACHE YES YES NO  YES

SQL>
```

33. 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>
```

34. 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 -
```

35. 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_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_1j8gg2m0ryqvpdc5f94e5 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_1j8gg2m0ryqvp08763816 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 1j8gg2m0ryqvpa496391d MANUAL-LOAD    YES YES NO  YES

SQL>
```

36. 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: 34974602

--------------------------------------------------------------------------------------------
---------------
| Id  | Operation                          | Name           |
--------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |                 |
|   1 |  HASH GROUP BY                     |                 |
|   2 |   NESTED LOOPS                     |                 |
|   3 |    PARTITION RANGE ALL             |                 |
|   4 |     TABLE ACCESS BY LOCAL INDEX
                              ROWID BATCHED| 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_1j8gg2m0ryqvp08763816" used for
this statement

19 rows selected.
```

37. 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_18a1ef14c17f5b75');

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
```

Workshop 1

Workshop 1: Overview

Workshop Overview

In this workshop, you will perform the following tasks:

- 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 the `hr` schema.

The connection details are as follows:

Connection Name: hr_connection

Username: hr

Password: hr

Hostname:

localhostPort:

1521

Service name: orclpdb

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
/
DESCRIBE HR.DEPT
/
DROP INDEX hr.dept_dept_id_idx1
/
CREATE INDEX hr.dept_dept_id_idx1
ON hr.dept (dept_id)
NOLOGGING
COMPUTE STATISTICS
/
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 the 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 w1_s1_e.sql, or the SQL Developer EXPLAIN PLAN tool icon in the Worksheet to gather the execution

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 the **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);
```

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.

Answer:

1) No, the index `DEPT_DEPT_ID_IDX2` is not used.

2) There was a data type mismatch. `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?

      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 subsequent`

Workshop 2

Workshop 2: Overview

Workshop Overview

In this workshop, you will perform the following tasks:

- 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

1. Display an execution plan using the `EXPLAIN PLAN` command.

   a. Connect to the `hr` schema in SQL Developer.

   b. 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)
/
```

   c. 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);
```

Answer:

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=:empid
/
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 the execution plan of any loaded cursor?
3) Was the `EMP_ID_IDX` index used?

Answer:

1) `0vy06dgmbqh9q SQL_ID`
2) `dbms_xplan.display_cursor` is used to format and display the contents of the 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, the 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

Workshop 3: Overview

Workshop Overview

In this 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:
   localhostService
   name: orclpdb

Note: The scripts used in this workshop are available under the workshops folder located in
/home/oracle/labs/workshops.

Make sure the sh user has the following privilege:

```
GRANT DBA to sh;
```

If not, connect as the **sys** user in SQL Developer and grant the privilege to sh.

b. Execute the **`setup03.sql`** to set up the environment for this workshop.

```
DROP TABLE sh.country PURGE
/

CREATE TABLE sh.country (
    country_id number primary key,
    country_name varchar(40),
    country_subregion varchar(30),
    country_region varchar2(20)
)
/

INSERT INTO sh.country
SELECT country_id, country_name, country_subregion,
country_region
FROM  sh.countries
/

commit
/

CREATE INDEX sh.count_country_region_idx
ON sh.country(country_region)
NOLOGGING
COMPUTE STATISTICS
/

exec
dbms_stats.gather_table_stats('SH','COUNTRY',method_opt=>'for
all indexed columns',cascade=>true)
/

DESC sh.country
/
```

c. Review the `ws03.sql` file.

d. Gather the execution plan of the SQL statement in the `ws03.sql` file using the `EXPLAIN PLAN` command given as follows or you can use `w3_s1_c.sql` file.

```
EXPLAIN PLAN FOR SELECT country_name, country_subregion
FROM sh.country
WHERE country_region = 'Europe'
```

```
ORDER BY country_name
/

SELECT * FROM TABLE(dbms_xplan.display);
```

1) What is the name of the index used?
2) What is the total cost?

Answer:

1) The `COUNT_COUNTRY_REGION_IDX` index is used.
2) Cost is 3.

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, `COUNT_CN_CSR_IDX` or you can use the `w3_s2_a.sql` file.

Note: `Index order: country_name, country_subregion`

```
DROP INDEX sh.count_cn_csr_idx
/
CREATE INDEX sh.count_cn_csr_idx
ON sh.country (country_name, country_subregion)
/
```

   b. Gather and analyze the execution plan of the SQL statement again by using the `EXPLAIN PLAN` command or you can use the `w3_s2_b.sql` file.

```
EXPLAIN PLAN FOR SELECT country_name, country_subregion
FROM sh.country
WHERE country_region = 'Europe'
ORDER BY country_name, country_subregion
/

SELECT * FROM TABLE(dbms_xplan.display);
```

1) Was the `COUNT_CN_CSR_IDX` index used? If not, add an index hint to allow the optimizer to use the `COUNT_CN_CSR_IDX` index and gather the execution plan by using the `EXPLAIN PLAN` command. You can use `w3_s2_b1.sql`.

```
EXPLAIN PLAN FOR SELECT /*+ index (count_cn_csr_idx) */
country_name, country_subregion
FROM sh.country
WHERE country_region = 'Europe'
ORDER BY country_name, country_subregion
/

SELECT * FROM TABLE(dbms_xplan.display);
```

2) Was a sort operation avoided?
3) Was the cost of the plan better than the one in Task 1?

Note: `sh.count_cn_csr_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.country MODIFY (country_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.`

3. Examine the indexed `ORDER BY` columns.

   a. Use the sample code to create another composite index, `COUNT_CSR_CN_IDX`, on the `ORDER BY` columns or you can use the `w3_s3_a.sql` file.

Note: `Index order: country_subregion, country_name`

```
DROP INDEX sh.count_csr_cn_idx
/
CREATE INDEX sh.count_csr_cn_idx
ON sh.country (country_subregion, country_name)
/
```

   b. Gather and analyze the execution plan of the SQL statement again by using the `EXPLAIN PLAN` command. You can use the `w3_s3_b.sql` file.

```
EXPLAIN PLAN FOR SELECT country_name, country_subregion
FROM sh.country
WHERE country_region = 'Europe'
ORDER BY country_name, country_subregion
/
SELECT * FROM TABLE (dbms_xplan.display);
```

1) Was the `COUNT_CSR_CN_IDX` index used? If not, add an index hint to allow the optimizer to use the `count_csr_cn_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 (count_csr_cn_idx) */
country_name, country_subregion
FROM sh.country
WHERE country_region = 'Europe'
ORDER BY country_subregion, country_name
/
SELECT * FROM TABLE (dbms_xplan.display);
```

2) Was a sort operation avoided?

3) Was the cost of the plan better than the one in Task 1?

Note: The COUNT_CSR_CN_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 following questions based on your observation.

a. What do you observe?

b. Execute the `cleanup03.sql` script for the next workshop.

```
DROP INDEX sh.count_country_region_idx
/
DROP INDEX sh.count_cn_csr_idx
/
DROP INDEX sh.count_csr_cn_idx
/
DROP TABLE sh.country PURGE
/
```

5. When using the indexed `ORDER BY` column to avoid a sort operation, you should consider the following:
   a. All the `order by` columns should be in the composite index.
   b. `ORDER BY` columns should be of the same order as those 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`

Workshop 4

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

```
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);
```

`w4_s1_d.sql` file.

1) Was the `TIME_IDX` index used?
2) What is the total cost?

Answer:

1) The `TIME_IDX` index is not used.
2) Cost is 511 in this case.

Note: `In this case, the cost is 511. 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?

setup04.sql | ws04.sql | w4_s1_d.sql | w4_s2_a.sql | w4_s2_b.sql

SQL Worksheet | History

0.072 seconds | sh_connection

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 = '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);
```

Script Output | Task completed in 0.072 seconds

```
Explained.


PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------------------------------------------
Plan hash value: 41534641

-------------------------------------------------------------------------------------------------
| Id  | Operation                              | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                       |          |     2 |    88 |   175   (1)| 00:00:01 |
|   1 |  SORT GROUP BY                         |          |     2 |    88 |   175   (1)| 00:00:01 |
|*  2 |   HASH JOIN                            |          |   551 | 24244 |   174   (0)| 00:00:01 |
|*  3 |    TABLE ACCESS FULL                   | PROD1    |    14 |   294 |     2   (0)| 00:00:01 |
|   4 |    TABLE ACCESS BY INDEX ROWID BATCHED| SALES1   |  2754 | 63342 |   172   (0)| 00:00:01 |
|*  5 |     INDEX RANGE SCAN                   | TIME_IDX |  2754 |       |    13   (0)| 00:00:01 |

PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------

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

   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')

19 rows selected.
```

Answer:

1) The `TIME_IDX` index is used.

Note: `Redesigning 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?`

Workshop 5

Workshop 5: Overview

Workshop Overview

In this workshop, you will perform the following tasks:

- 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 CompositeIndex

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. Set up.
   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 the 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?

Answer:

1) The `CUST_INCOME_STATE_IDX` index is used.
2) The cost is 202.

Note: `The cost is 202 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?

Answer:

1) The `CUST_INCOME_STATE_IDX` index is used.
2) The cost is 53.

Note: `The cost is 53 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, 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?

Answer:

1) The `CUST_INCOME_STATE_IDX` index is used.
2) The cost is 154.

Note: `The cost is 154 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';
```

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)
/
```

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?

Answer:

1) The `CUST_STATE_INCOME_IDX` index is used.
2) The cost is 37.

Note: `The cost is 37 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`

Workshop 6

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 SQLStatement

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 loop 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)| 00:00:01 |
|   1 |  NESTED LOOPS                |         |       |       |            |          |
|   2 |   NESTED LOOPS               |         |     1 |    22 |     4   (0)| 00:00:01 |
|*  3 |    TABLE ACCESS FULL         | EMP     |     1 |     9 |     3   (0)| 00:00:01 |
|*  4 |    INDEX UNIQUE SCAN         | PK_DEPT |     1 |       |     0   (0)| 00:00:01 |
|   5 |   TABLE ACCESS BY INDEX ROWID| DEPT    |     1 |    13 |     1   (0)| 00:00:01 |
```

Tasks

1. 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 = I/O 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:
   a. sreadtim (db_block_size: 8K): (_____)
   b. 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)
```

```
sreadtim = ioseektim + db_block_size / iotfrspeed
       = 10ms + 8096/4096 = 12 ms
CPU Costing Factor = cpuspeed * 1000 * sreadtim
                = 1999 * 1000 * 12 = 23988000
CPU Cost Scale Factor = 1 / CPU Costing Factor
                  = 1/(1999 * 1000 * 12) =
                  0.000000417
(nearly zero)
```

3. Review the following section for the outer table. Where is the outer table cost 3 from?
   a. Outer Table Cost (__3__) = 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
```

```
Outer Table Cost = IO Cost + CPU Cost
              = Resp_io + (Resp_cpu * CPU Cost Scale Factor)
              = 3 + (39507 * 0.000000417)
              = 3 + 0.0162 (nearly zero)
```

4. Review the following section that includes the inner table information.
   a. Outer Table Card: (_____)
   b. Inner Table Cost (__1___) = 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  resc_cpu: 47849
Join Card:  1.000000 = = outer (1.000000) * inner (4.000000) * sel (0.250000)
Join Card - Rounded: 1 Computed: 1.00
```

```
Inner Table Cost = IO Cost + CPU Cost
             = resc_io + (resc_cpu * CPU Cost Scale Factor)
             = 1 + (8341 * 0.000000417)
             = 1 + 0.0162 (nearly zero)
```

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 loop joins are TRUE?
   a. The optimizer uses nested loop 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

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 sec = Plan Cost * sreadtim = 962 * 12ms

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 preceding 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
```

Notes

- 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.

6. 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`

7. 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.
   a. What is the computed elapsed time based on the cost of 4000?
   b. Was the Oracle Optimizer estimated accurately?
      1) Yes
      2) No
   c. What could be the possible reasons for an inaccurate cost?

Answer:

a. Computed elapsed time = Plan Cost * sreadtim (12 ms)

```
= 4000*12(ms)
= 48000 ms
```

b. No

c. The reasons for inaccurate cost are:
   1) Large `IN` list
   2) 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

Workshop 8

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 preceding output of `TKPROF` 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 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 the `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.

	COUNTRY_ID	COUNT(*)	PCT
1	52769	597	1.0756756756756756756756756756756756756756
2	52770	7780	14.0180180180180180180180180180180180180180
3	52771	712	1.2828828828828828828828828828828828828828
4	52772	2010	3.6216216216216216216216216216216216216216
5	52773	403	0.7261261261261261261261261261261261261261
6	52774	831	1.4972972972972972972972972972972972972973
7	52775	832	1.4990990990990990990990990990990990990991
8	52776	8173	14.7261261261261261261261261261261261261613
9	52777	383	0.6900900900900900900900900900900900900900901
10	52778	2039	3.6738738738738738738738738738738738738738738
11	52779	3833	6.9063063063063063063063063063063063063063063
12	52782	624	1.1243243243243243243243243243243243243243243
13	52785	244	0.4396396396396396396396396396396396396396396
14	52786	708	1.2756756756756756756756756756756756756756756
15	52787	75	0.1351351351351351351351351351351351351351351
16	52788	91	0.1639639639639639639639639639639639639639639
17	52789	7557	13.6162162162162162162162162162162162162162
18	52790	18520	33.3693693693693693693693693693693693693693693
19	52791	88	0.1585585585585585585585585585585585585585586

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'
/
```

Script Output | Query Result

SQL | All Rows Fetched: 23 in 0.257 seconds

	COLUMN_NAME	HISTOGRAM	NUM_BUCKETS
1	CUST_LAST_NAME	HYBRID	254
2	CUST_GENDER	NONE	1
3	CUST_YEAR_OF_BIRTH	NONE	1
4	CUST_MARITAL_STATUS	NONE	1
5	CUST_STREET_ADDRESS	NONE	1
6	CUST_POSTAL_CODE	NONE	1
7	CUST_CITY	NONE	1
8	CUST_CITY_ID	NONE	1
9	CUST_STATE_PROVINCE	FREQUENCY	145
10	CUST_STATE_PROVINCE_ID	NONE	1
11	COUNTRY_ID	FREQUENCY	19
12	CUST_MAIN_PHONE_NUMBER	NONE	1
13	CUST_INCOME_LEVEL	NONE	1
14	CUST_CREDIT_LIMIT	NONE	1
15	CUST_EMAIL	NONE	1
16	CUST_TOTAL	NONE	1
17	CUST_TOTAL_ID	NONE	1
18	CUST_SRC_ID	NONE	0
19	CUST_EFF_FROM	NONE	1
20	CUST_EFF_TO	NONE	0
21	CUST_VALID	NONE	1
22	CUST_ID	HYBRID	254
23	CUST_FIRST_NAME	NONE	1

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)
/
```

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);
```

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);
```

ws08.sql | w8_s2_g.sql

SQL Worksheet | History | 0.484 seconds | sh_connection

Worksheet | Query Builder

```
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);
```

Script Output | Task completed in 0.484 seconds

```
2

Plan hash value: 2854028564

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

PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                       |                      |       |       |   289 (100)|          |
|   1 |  SORT ORDER BY                         |                      |    71 |  2130 |   289   (4)| 00:00:01 |
|   2 |   HASH GROUP BY                        |                      |    71 |  2130 |   289   (4)| 00:00:01 |
|*  3 |    HASH JOIN                           |                      |  9792 |  286K|   286   (3)| 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                |                      |  921K| 9000K|   266   (3)| 00:00:01 |
|   7 |      TABLE ACCESS FULL                 | SALES                |  921K| 9000K|   266   (3)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):

PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------------------------------------------------
---------------------------------------------------

   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.

h. Try to explain why the execution plans are different in step f and g.
   1) Why are the plans different? Does the optimizer consider two similar SQL statements in step f and g as the same SQL statements?
   2) How many hard parses occurred?

i. 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 the `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: fenago

      Role: SYSDBA

      Host Name: localhost

Service name: orclpdb

```
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)
/
```

w8_s3_a.sql | w8_s3_c.sql | w8_s3_c1.sql

SQL Worksheet | History

0.43200001 seconds | sh_connection

Worksheet | Query Builder

```
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)
/
```

Script Output

Task completed in 0.432 seconds

```
233 rows selected.


PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------------------
SQL_ID  bku7bn21mqnjr, 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

Plan hash value: 1733060975

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

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |           |       |       |   499 (100)|          |       |       |
|   1 |  SORT ORDER BY           |           |   620 | 23560 |   499   (6)| 00:00:01 |       |       |
|   2 |   HASH GROUP BY          |           |   620 | 23560 |   499   (6)| 00:00:01 |       |       |
|*  3 |    HASH JOIN             |           |  7059 |   261K|   497   (6)| 00:00:01 |       |       |
|   4 |     VIEW                 | VW_GBC_5  |  7059 |   124K|   286  (10)| 00:00:01 |       |       |
|   5 |      HASH GROUP BY       |           |  7059 | 70590 |   286  (10)| 00:00:01 |       |       |
|   6 |       PARTITION RANGE ALL|           |   921K|  9000K|   266   (3)| 00:00:01 |     1 |    28 |
|   7 |        TABLE ACCESS FULL | SALES     |   921K|  9000K|   266   (3)| 00:00:01 |     1 |    28 |
|*  8 |     TABLE ACCESS FULL    | CUSTOMERS | 18520 |   361K|   211   (1)| 00:00:01 |       |       |
-----------------------------------------------------------------------------------------------------------


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

w8_s3_a.sql | w8_s3_c.sql | w8_s3_c1.sql

SQL Worksheet | History

0.23199999 seconds — sh_connection

Worksheet | Query Builder

```
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)
/
```

Script Output

Task completed in 0.232 seconds

```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------------------
SQL_ID  c1yprd639fkh7, 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     | Pst
------------------------------------------------------------------------------------------------------------------------

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                       |                     |       |       |   289 (100)|          |
|   1 |  SORT ORDER BY                         |                     |    71 |  2130 |   289   (4)| 00:00:01 |
|   2 |   HASH GROUP BY                        |                     |    71 |  2130 |   289   (4)| 00:00:01 |
|*  3 |    HASH JOIN                           |                     |  9792 |  286K|   286   (3)| 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                |                     |  921K| 9000K|   266   (3)| 00:00:01 |
|   7 |      TABLE ACCESS FULL                 | SALES               |  921K| 9000K|   266   (3)| 00:00:01 |
------------------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------

   3 - access("S"."CUST_ID"="C"."CUST_ID")
   5 - access("COUNTRY_ID"=52787)

Note
```

d. Drop the `set_cursor_sharing` trigger by connecting as user `sys`. Use the `w8_s3_d.sql file.`

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.`

   a. Connect to the `sh` schema in SQL Developer.

   b. 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)
/
```

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)
/
```

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.

Note: `Copy SQL_ID and Plan_hash_value values( for w8_s5_a.sql and w8_s5_a1.sql) from the outputs of step 4(b) and 4(c).`

```
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 %';
```

Workshop 9

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 ExecutionPlan

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')
/
alter session set optimizer_use_sql_plan_baselines = true
/
alter session set optimizer_capture_sql_plan_baselines = true;
```

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 also use the `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 a better plan, you can associate the original SQL with the new

```
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)%';
```

plan (hinted plan).

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>>'));
```

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>>');
```

d. Check the loaded plan in SQL Plan Baseline by using the following sample code, or you can use the `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)%';
```

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)%';
```

   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('gnmzkah3xhsbr'));
```

```
PLAN_TABLE_OUTPUT
SQL_ID  gnmzkah3xhsbr, child number 0
-------------------------------------
SELECT /*+INDEX(cust cust_yob_bix) */ * FROM sh.cust WHERE
cust_year_of_birth < 70 AND cust_gender = 'M'

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       |              |       |       |            |          |
|*  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: 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>>');
```

```
var res number;
exec :res := dbms_spm.load_plans_from_cursor_cache(sql_id=> 'gnmzkah3xhsbr', plan_hash_value=> '1264022301' , sql_handle=> 'SQL_bd38613f52934cc0');
```

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

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%';
```

Note: The first 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>>');
```

```
select sql_text, sql_handle, plan_name, enabled, accepted
from dba_sql_plan_baselines
where sql_text like 'SELECT /*+ INDEX_COMBINE(cust%';
```

5. Test the SQL statement with the loaded plan.

   a. Clean up the shared pool for testing in the next step. You can use the `w9_s5_a.sql` file. Use the sys user to execute this statement.

```
ALTER SYSTEM FLUSH SHARED_POOL;
```

b. 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);
```

--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'
/
select sql_id, sql_plan_baseline from v$sqlarea where sql_text
like 'SELECT /*+ INDEX_COMBINE(cust%';
```

--Option 3:

```
SET LINESIZE 150
SET PAGESIZE 2000
SELECT t.* FROM TABLE (DBMS_XPLAN.DISPLAY_SQL_PLAN_BASELINE ('<<
Please enter SQL_HANDLE value>>'))t;
```

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
/
```